Escuela Politécnica Superior
Departamento de Ingeniería Informática

DESIGN AND IMPLEMENTATION OF SECURE PROTOCOLS FOR PRACTICAL AUTHENTICATION AND FAIR ANONYMITY SYSTEMS

PHD DISSERTATION

Jesus Diaz Vico

Madrid, April 2015

Escuela Politécnica Superior
Departamento de Ingeniería Informática

DESIGN AND IMPLEMENTATION OF SECURE PROTOCOLS FOR PRACTICAL AUTHENTICATION AND FAIR ANONYMITY SYSTEMS

PHD DISSERTATION

Jesus Diaz Vico

Madrid, April 2015

ADVISORS: Francisco de Borja Rodríguez Ortiz
David Arroyo Guardeño

Committee named by the Hon. Rector of the Universidad Autónoma de Madrid as of _____, 2015.

President: Dr. _____

Secretary: Dr. _____

Member: Dr. _____

Member: Dr. _____

Member: Dr. _____

Thesis defense held at the Universidad Autónoma de Madrid as of _____, 2015.

Examiner: Dr. Alvaro Ortigosa Juárez

External examiner: Dr. Angelos D. Keromytis

External examiner: Dr. Shujun Li

The PRESIDENT The SECRETARY

The MEMBERS

Acknowledgements

In the long (long...) time necessary to complete this thesis, there have consequently been many people to whom I owe gratitude. Starting of course with Paco and David who, since the beginning never doubted me and have been a constant support by all possible means and in every possible situation that has arised (which grants them an inexplicable merit, given the current societal and cultural situation). Angelos, Seung Geol and Moti, who did not hesitate in figuring out something in which we could work together during my stay at Columbia. Their always wise words have been invaluable, and the experience they helped me gain is something that I could not even imagine a few years ago.

My parents, brother and Eva, who have supported me and borned me even when not even I could do it. And not just during the thesis, but much more time before. My friends and family, who have had to get along with not seeing me much for a long time, and Edzer, who treated me as family while at New York. My lab mates at the UAM, my former coworkers at the CritpoLab and my current coworkers at INCIBE, all of them have been a foothold during this time.

This thesis somehow belongs to all these people (and probably more) too. Thanks to them, I can finally start gazing at what is next. This has been just the beginning.

Agradecimientos

En el largo (largo...) tiempo necesario para completar esta tesis, son muchas las personas a las que debo gratitud. Empezando por supuesto por Paco y David, quienes no dudaron de mí desde el principio y han sido un apoyo constante, por todos los medios posibles (lo cual es un mérito más allá de lo explicable, dada la coyuntura social y cultural actual). Angelos, Seung Geol y Moti, quienes no dudaron en buscar algo en lo que pudiéramos trabajar juntos durante mi estancia en Columbia. Sus siempre sabias palabras han sido invaluables, y la experiencia que me han ayudado a vivir es algo que no habría imaginado hace unos años.

Mis padres, hermano y Eva, que me han apoyado y soportado incluso cuando ni yo mismo podía hacerlo. Y no sólo durante la tesis, sino mucho antes. Mis amigos y mi familia, que han tenido que conformarse con verme muy poco durante mucho tiempo, y Edzer, que me trató como si fuera familia mientras estuve en Nueva York. Mis compañeros de laboratorio en la UAM, mis antiguos compañeros en el CriptoLab y mis actuales compañeros en INCIBE, todos ellos han sido un importante punto de apoyo durante todo este tiempo.

Esta tesis es, de alguna forma, también de todas estas personas (y probablemente más). Gracias a ellos, por fin podré saber qué viene a continuación. Esto ha sido sólo el principio.

Abstract

With the huge growth of information and communication systems, as well as the computing power, privacy has become a main concern for Internet users. Certainly, nowadays users tend to prefer privacy respectful systems and, consequently, companies providing software solutions also need to worry about it. Nevertheless, the privacy provided by current systems many times reduces to the need of placing too much trust into legal protections. Conversely, the contributions by the research community in this direction many times fail to produce realistic enough solutions, hardly flexible, scalable or deployable in current systems, and thus, impractical.

In this thesis, we attempt to bridge this gap between the practical but barely robust systems in the "real world" and the robust but barely practical ones of the "academic counterpart". Specifically, we base our proposals in currently deployed protocols and systems, but extend them for making them suitable to implement privacy, mainly through fair anonymity. Moreover, our approach for incorporating privacy-by-design grows from addressing less complex tasks towards tackling more complex issues based on the composition of the simple ones. This also allows us to establish a flexible framework from which solutions applicable for contexts other than those explored here may be derived. In turn, this helps to reduce the complexity of deploying new systems from scratch which, as stated, is our initial objective.

In more detail, in order to ease the design and deployment of privacy respectful systems, we proceed as follows. We first propose a methodology for designing protocols and systems and verifying that they meet the required security properties. This methodology is used to create and verify the protocols and systems proposed afterwards. On the other hand, since we make important use of group signatures for providing privacy through anonymity, and we aim to ease the costs of deploying new systems, we describe an extensible C library that we have implemented and released in an alpha stage, offering a unified API for group signatures. Subsequently, we make use of these global building blocks for creating technology that would most probably be necessary in every privacy respectful system. Specifically, given that the initial problem in any online platform requiring personalized interaction or some kind of authentication is to actually distribute digital identities, we propose SEBIA, a protocol based on EBIA (the typical email-based registration system) that ensures a reasonable level of security for many contexts. Specifically, it allows the distribution of anonymous identities like the ones that are used as a base to create privacy systems in subsequent chapters, and that are based in group signatures. Once having addressed the distribution problem, we extend the widely deployed X.509 PKI in order for it to be suitable for managing anonymous identities. Specifically, we propose extensions to the OCSP and CRL mechanisms, and create a new X.509-like protocol for communicating evidences of misbehavior (which, regrettably, is a problem sometimes derived from anonymity). With this contributions, we allow the creation of advanced privacy respectful systems based on

anonymity. In fact, with the aim of showing it, we design two systems. First, a comprehensive online shopping system that allows anonymous purchases while being also compatible with typical e-commerce benefits, like customer-specific marketing techniques. Secondly, we define an extension to the Tor network which, also based on the same mechanisms for managing anonymity, would allow to shift from full anonymity to fair anonymity.

Moreover, for several of the proposals made in this thesis, we have implemented actual prototypes that have enabled us to perform initial profiling tasks. Despite being preliminary versions lacking optimization, the results indicate that our proposals incur in acceptable costs.

Resumen

Con el gran auge de los sistemas de la información y las comunicaciones, junto con la capacidad de cómputo, los usuarios han empezado a preocuparse por su privacidad. Por ello, cada vez prefieren más los sistemas que son respetuosos con su información personal, lo cual está llevando a las compañías desarrolladoras de software a preocuparse también por la privacidad de los usuarios. No obstante, las garantías de privacidad en los sistemas actuales normalmente se reducen a mecanismos de protección legal, en los que los usuarios deben confiar. Por el contrario, las contribuciones hechas desde la comunidad académica normalmente consisten en sistemas poco prácticos o realistas y poco adaptables a las infraestructuras actuales.

En esta tesis, intentamos reducir esta brecha entre los sistemas prácticos pero poco robustos del "mundo real" y los sistemas robustos pero poco prácticos del "mundo académico". Para ello, nos basamos en protocolos y sistemas actualmente utilizados en la industria, pero adaptándolos de forma que sean respetuosos con la privacidad a través de primitivas criptográficas avanzadas, proporcionando anonimato justo. En concreto, empezamos abordando tareas más sencillas para luego crear sistemas más complejos. Esto nos permite crear un marco de trabajo flexible, a partir del cual se pueden derivar soluciones aplicables a contextos distintos de los que aquí se muestran. Al mismo tiempo, esto ayuda a reducir la complejidad de desplegar nuevos sistemas desde cero, cumpliendo con nuestro objetivo.

Con algo más de detalle, para facilitar el diseño e implementación de sistemas respetuosos con la privacidad, procedemos de la siguiente manera. Primero, proponemos una metodología para diseñar protocolos y sistemas, verificando que cumplen los requisitos de seguridad establecidos. Esta metodología la utilizamos para crear y verificar los protocolos y sistemas propuestos más adelante. Por otro lado, dado que hacemos un uso importante de firmas grupales para proporcionar privacidad a través de anonimato, y nuestra intención es facilitar la creación de nuevos sistemas, presentamos una librería para firmas grupales, escrita en C. Esta librería, aún en fase alfa, es fácilmente extensible, de forma que se pueden añadir nuevos esquemas en caso de ser necesario, manteniendo una API unificada. A continuación, hacemos uso de estos pilares básicos para crear componentes tecnológicos que cualquier sistema respetuoso con la privacidad probablemente requerirá. En concreto, dado que en toda plataforma online la primera operación necesaria es registrarse en la misma, proponemos SEBIA, un protocolo basado en EBIA (el típico sistema de registro basado en emails), pero que proporciona un nivel de seguridad suficiente para muchos escenarios. Este protocolo, concretamente, permite la distribución de identidades digitales anónimas como las usadas en los siguientes capítulos y basadas en firmas grupales. De hecho, una vez distribuidas las identidades, es necesario disponer de mecanismos eficientes para gestionarlas. Para ello, extendemos la infraestructura de clave pública X.509 con el fin de adaptarla para la gestión de identidades anónimas. En concreto, extendemos los

mecanismos OCSP y CRL, además de crear un nuevo protocolo que, siguiendo los mismos principios de diseño de X.509, permite la distribución de evidencias de comportamientos ilegítimos (lo cual, desgraciadamente, es un problema que suele acompañar al anonimato). Con estas contribuciones, facilitamos la creación de sistemas avanzados y respetuosos con la privacidad, basados en el anonimato, y al mismo tiempo compatibles con tecnologías actuales. Para mostrarlo, diseñamos un sistema de compras online que permite la realización de compras anónimas y que es además compatible con las técnicas de marketing actuales. Además, proponemos una extensión para la red Tor que, también basándose en los mecanismos presentados, podría permitir la migración de dicha red a un sistema de anonimato justo.

Por último, para varias de las propuestas hechas en esta tesis, se han implementado prototipos que nos han permitido realizar un análisis inicial. A pesar de ser versiones preliminares sin optimizar, los resultados indican que nuestras propuestas introducen sobrecostes aceptables.

Contents

List of Figures

List of Tables

Part I

Introduction

CHAPTER 1

Thesis overview

In software engineering it is frequent to focus verification efforts in the search for *implementation bugs* aiming to validate a system or protocol. This kind of defects certainly have severe impact in the affected product, fact that has been corroborated by recent events, like the *Shellshock*[1] bug in Bash or the *Heartbleed*[2] vulnerability affecting OpenSSL. However, while looking for these bugs is necessary, these problems may generally be fixed by applying software patches to the vulnerable components.

In addition, *design flaws* are another type of defect that typically receives less attention from the computer science security community, when compared with their implementation counterpart. Quoting the 2014 IEEE Report "*Avoiding the top 10 software security design flaws*"[3] by the Center For Secure Design:

> *A bug is an implementation-level software problem. Bugs may exist in code but never be executed. A flaw, by contrast, is a problem at a deeper level. Flaws are often much more subtle than simply an off-by-one error in an array reference or use of an incorrect system call. A flaw might be instantiated in software code, but it is the result of a mistake or oversight at the design level.*

For instance, a typical design flaw may be just assuming that a certain entity involved in the system is trustworthy, when it is not. Such a improperly-trusted entity could endanger the whole system even in the remote case that no implementation bug exists, resulting for example in a massive information leak. According to the mentioned report, some of the top 10 security concepts that are typically ignored and lead to design flaws, are:

- "*Earn or give, but never assume, trust*".
  - That is, you should never just assume that some entity is trustworthy.
- "*Use an authentication mechanism that cannot be bypassed or tampered with*".
- "*Authorize after you authenticate*".
  - Specifically, an already authenticated user may not be authorized to perform certain operations.

[1] `http://seclists.org/oss-sec/2014/q3/649`. Last access: March 28th, 2015.

[2] `http://heartbleed.com/`. Last access: March 28th, 2015.

[3] `http://cybersecurity.ieee.org/center-for-secure-design.html`. Last access: March 28th, 2015.

- *"Use cryptography correctly"*.
- *"Identify sensitive data and how they should be handled"*.
- *"Always consider the users"*.
  - Users are almost always the weakest link in the security chain.

All these design principles, when not applied during the design stage, may be the cause of severe over-costs at the development stage (and even worse, at the production stage, when they may have even been exploited), as shown in Figure 1.1.

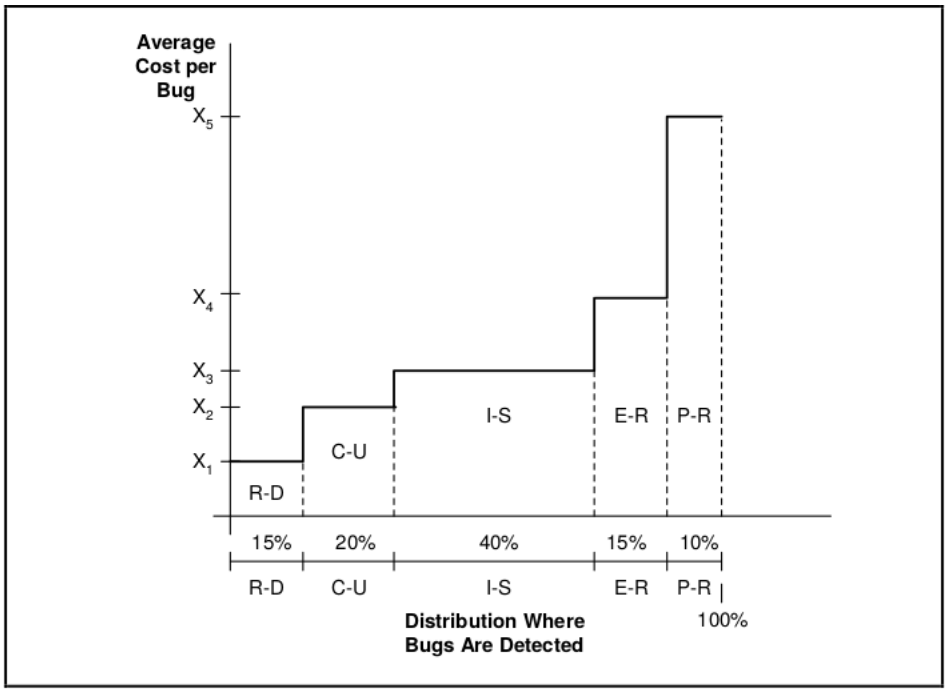


Figure 1.1: Costs of addressing software problems, depending on the stage in the Software Development Lifecycle in which they are applied. Source: [179].

This comes in line with the *security by design* practice. Indeed, in the same way than the application of an appropriate *Software Development Lifecycle (SDLC)* helps producing software of high quality, the application of *Security Development Lifecycle* helps producing secure software [168]. In turn, secure software means higher quality software. In this work we further develop the security by design concept towards the more recent trend *privacy by design*[4]. Privacy is indeed a complex matter that has many subtleties. On one hand, directly applying conventional computer security or cryptography methods does not automatically guarantee privacy. The typical misconception in this case

[4]See the *Privacy by Design* resolution, where privacy is recognized as an essential foundation upon which free societies are built, at `http://www.ipc.on.ca/site_documents/pbd-resolution.pdf` (last access on March 28th, 2015).

is assuming that just by applying encryption over private data actually ensures its privacy: but, for instance, the mere leakage of the existence of that encrypted data may actually violate someone's privacy. A representative example is that of a whistle-blower communicating with the police: if the criminal organization he belongs to realizes he is exchanging information with the police, even if it is encrypted, it will give him away. Another trending example is the current big data contributions by means of which, from what a priori many people consider "harmless" information, sometimes highly private knowledge may be extracted[5] [72]. For instance, click patterns, visited websites or session duration may be used to extrapolate lifestyle, consumption habits, political preferences, etc. Thus, privacy is a subject that needs to be dealt with in detail, many times analyzing specifics of the application domain, but for which also general infrastructures endorsing it are still required. The complexity of achieving privacy is captured for the specific context of anonymous communications by the Tor infographic shown in Figure 1.2 [6]. There, it is emphasized that even when applying encryption (through HTTPS), privacy may still be violated.

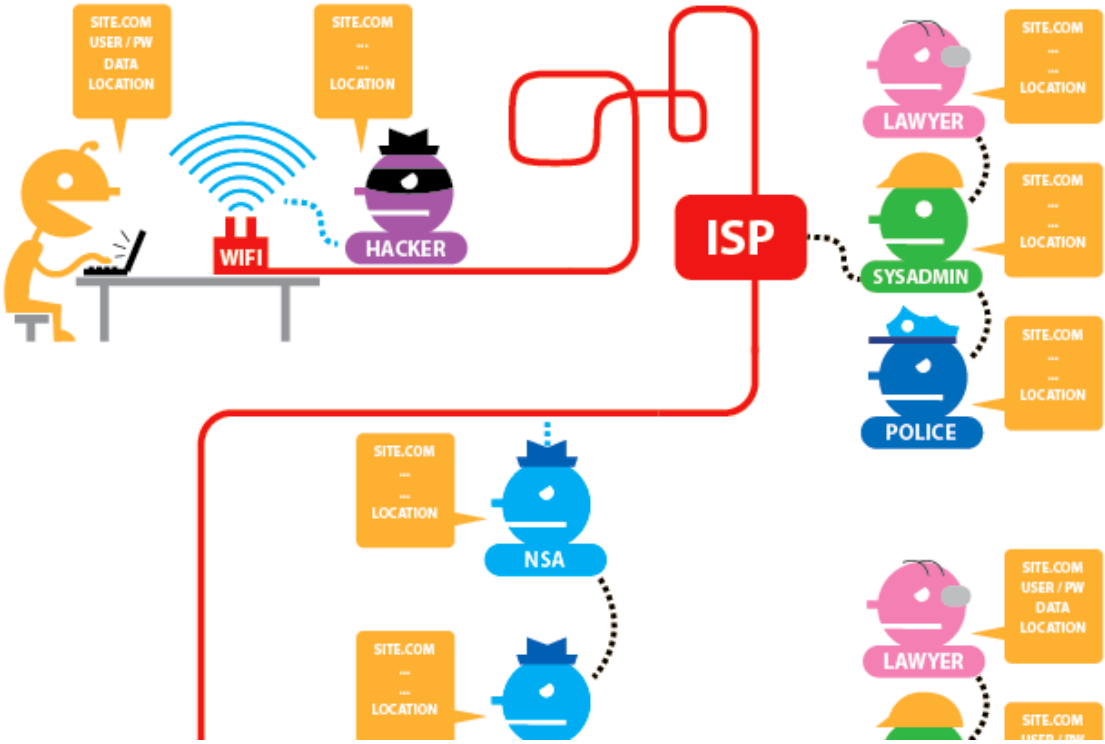


Figure 1.2: Confidentiality does not necessarily imply privacy. In web browsing, even when providing confidentiality (through the encryption of HTTPS), external actors may break the users' privacy. In the image, hackers, police, NSA, etc. are shown as being able to learn, for instance the URL visited by the user (site.com) and the user's location. Fragment of Tor infographic (source: EFF[6]).

On the other hand, applying too much privacy-enhancing technologies may not be desirable either. For instance, irrevocable anonymity may be misused as a protection

[5] http://www.stanfordlawreview.org/online/privacy-and-big-data/three-paradoxes-big-data. Last access on March 28th, 2015.

[6] https://www.eff.org/pages/tor-and-https. Last access on March 31st, 2015.

for performing dishonest actions despite the fact that is is certainly a primary tool for complying legitimate privacy purposes (like the whistle-blower example).

Another important principle in engineering is that a solution that nobody uses provides no real benefit, despite the many advantages it may provide. Thus, following the common sense and coming back to the security design principles, we may enunciate a general *pragmatism principle*, aimed to achieve a high acceptance rate for new systems:

> If some security property or functionality may be satisfactorily achieved through currently existing and widely deployed, accepted and tested technologies, then developing new ones should be avoided. Else, if modification is unavoidable, the impact on both the end users and the technology itself should be kept to a minimum, and the modifications should follow the security design principles. If new technology is really needed, then its design should follow the security design principles.

This does not mean that, when necessary, new systems may not be created, but that in order to design them, we should rely as much as possible on existing technologies and theories [98]. In a humorous manner, the vignette in Figure 1.3[7] from the webcomic XKCD captures one of the problems of not following what we have called the *pragmatism principle*.

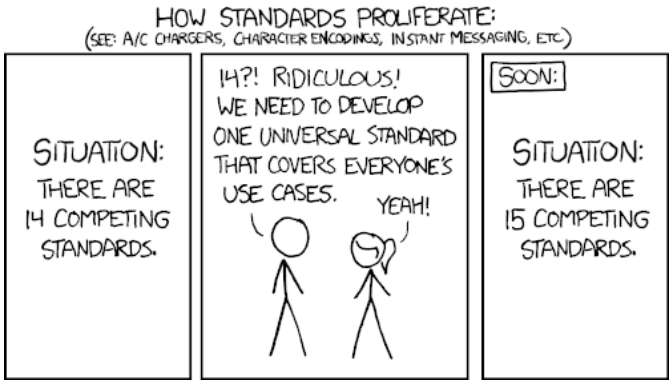


Figure 1.3: XKCD[7] *Standards* vignette .

In turn, by promoting privacy respectful systems by design through technology directly compatible with current infrastructures, usability would also be improved through a simplification of the resulting systems. This is an important factor to keep in mind, since usability has been a historical drawback in security systems and software [25, 183].

This is a relevant subject if we look into systems proposed in the academia. As we will see throughout this work, the proposals originated at the academic world, despite providing very interesting and robust security and privacy features, many times fail at being suitable for the current technological context. For instance, by requiring extensive modifications into the communications model (introducing unbearable costs) or requiring the addition of new entities that are hard to construct or emulate in real systems. Specific examples will be given in the related work sections of Chapter 7 and Chapter 8.

[7] `http://xkcd.com/927/`. Last access on March 28th, 2015.

In this work, we adhere to these guidelines for enabling the creation of *practical, privacy respectful and secure by design* systems through the construction of a complete framework allowing the distribution and management of private identities and infrastructures suitable for many contexts, using fair anonymity as a core privacy-endorsing component and based on currently existing technologies. A complete instantiation of this framework is described through an e-commerce setting, which exemplifies the subtle issues that need to be considered when creating application-specific privacy respectful systems. Moreover, to further illustrate the flexibility and practicality of our framework, we apply a similar construction for defining an extension to Tor [91], the most popular anonymizing network.

1.1 Objectives

In general lines, we aim to ease the task of creating privacy respectful but realistic systems through the application of cryptographic primitives, using existing infrastructures as much as possible. For achieving this, we identify the following key elements:

1. An intuitive design methodology that helps extracting formal security requirements from informal ones, enabling their verification.
2. Facilitate the complex task of actually implementing privacy respectful systems based cryptographic primitives for fair anonymity.

When addressed this issues, we would have eased the task of designing and implementing new systems and verifying their security in an intuitive manner. With this, our next task would be to actually create a framework for creating these systems. Specifically, in the context of privacy respectful systems, where privacy is achieved through anonymity, we identify the following core components:

3. A means for a secure, yet usable, distribution of identities in an online scenario.
4. Flexible enough mechanisms for managing the kind of identities that would be required in a privacy respectful settings.

Indeed, in the context of online applications, once generated the identities, we need to be able to distribute it with security guarantees adequate to the application. Further, managing anonymity is a complicated issue, but it is key for producing systems suitable in the real world. With the ability of distributing and managing anonymous identities, we attempt to:

5. Create systems that are compatible with the previous mechanisms.

Moreover, our focus will be in creating systems that are as much compatible as possible with the current infrastructures and technologies, with the aim of facilitating their deployment in the real world.

1.2 Outline

We first dedicate some pages for introducing related work in Chapter 2, which also explains some important concepts and definitions that will be used afterwards. The central chapters are distributed in three main parts.

The first part, spanning Chapter 3 and Chapter 4 contains two components that have been used as a base for the works presented in the subsequent parts. Specifically, in Chapter 3, we introduce the methodology that we have used for designing protocols and systems and verifying the security properties that they aim to achieve. Applications of this methodology to existing protocols and systems are included in order to show its functioning. Moreover, it is also applied to the protocol in Chapter 5 and the system in Chapter 7. Rather than committing to a specific theory or technology, this methodology proposes a (cyclic) sequence of tasks to perform in order to achieve the desired security level and properties. In Chapter 4 we outline a C library for group signatures, providing a unified API and extensibility through the addition of new schemes. This library has been used extensively during the subsequent chapters, in order to implement prototypes of the protocols and systems therein proposed.

In the second part, we describe the concrete protocols designed during this thesis. In Chapter 5 we propose a registration protocol for online systems, based on the typical email-based registration protocol [102], but improving its security properties while keeping its usability. Moreover, the security properties of this protocol are verified using the methodology proposed in Chapter 3. This protocol is a suitable alternative for distributing anonymous identities that may be employed in privacy respectful systems. However, while anonymity is probably the most used means to achieve privacy, there are (to the best of our knowledge) currently no solutions for effectively managing it. In order to fill this gap, we propose in Chapter 6 extensions to X.509 [150] that allow performing the main tasks for identity management (i.e. obtaining status information and revoking validity) over anonymous identities, plus adding support for new features intrinsic to them (like anonymity and unlinkability revocation).

These protocols allow an effective management of privacy through anonymous identities in more complex systems. In order to show it, in Part IV we describe an online shopping system (in Chapter 7) of independent interest, which using fair anonymity as a core, achieves a functionality set comparable to that of existing online shopping solutions, but having user-controlled privacy as its cornerstone. Finally, Chapter 8 proposes an extension to the Tor network for shifting from its full anonymity towards a fair anonymity model, preventing misuses of the network. Anew, this extension may benefit from the identity distribution and management protocols and mechanisms of previous chapters. Moreover, the cryptographic construction employed for this extension follows the same principles (based on group and blind signatures) than those used in the system designed in Chapter 7.

Additionally, in order to show feasibility in terms of additional costs, we include performance analysis based on initial prototypes that we have implemented for several of our protocols and systems.

To graphically summarize what we will see in the next chapters, Figure 1.4 depicts the relationship between the different contributions included in this thesis.

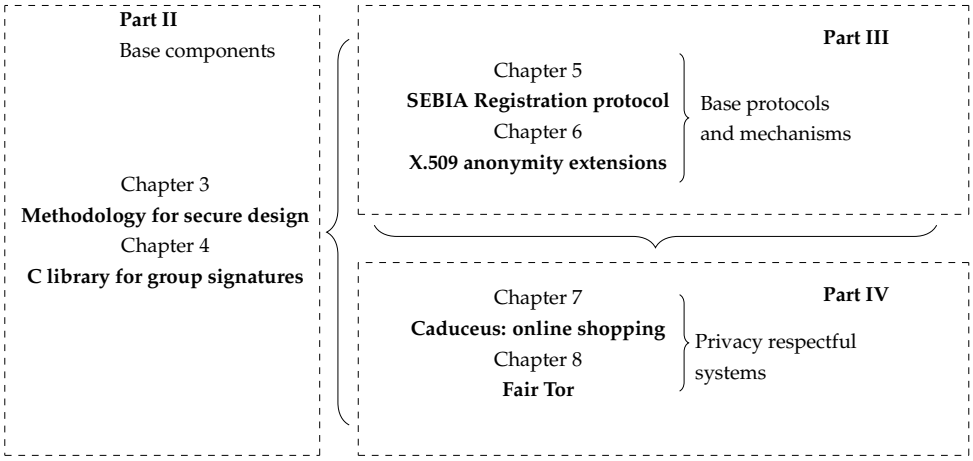


Figure 1.4: Diagram of relations between the different contributions of this work.

CHAPTER 2

Background

In this chapter, we make a summary of general concepts for computer security and applied cryptography systems, and define the notation used throughout this thesis. Additionally, we introduce the main tools and building blocks used as a base for our proposals. Namely, we introduce security verification techniques, focusing on formal methods for automated verification that will be applied for checking the security of our proposals. Subsequently, we introduce group and blind signatures, as well as other important primitives that are employed as core cryptographic primitives for building privacy respectful protocols and systems.

2.1 Security properties

Next, we give informal definitions of several important security properties which are key for understanding and reaching secure and privacy respectful systems and protocols.

While the properties within the CIA triad (Figure 2.1) are typically the base for any security system, many times it is necessary to use more specific definitions. In this work, we mainly refer to the properties defined below. Note however that precise definitions of these same properties may vary depending on the source. For instance, in [147], confidentiality and privacy are considered equivalent, and the base CIA triad includes authenticity within integrity.

Definition 1 (Confidentiality)**.** *Means that the information will be learned only by those who are authorized to do so.*

Definition 2 (Integrity)**.** *Ensures that the information is not altered, either consciously (and perhaps maliciously) or accidentally.*

Definition 3 (Authenticity)**.** *Guarantees the legitimacy of the information. It can be further divided in entity authentication, i.e., that an entity is who it says it is; and data origin authentication, i.e., that the information comes from the expected source.*

When dealing with privacy issues, we first find that *privacy* itself is a more abstract property than the ones above, since its requirements vary depending on the specific context. Many times it suffices with guaranteeing confidentiality of the information being transmitted. In other contexts, leaking that Alice is talking to Bob is in itself a privacy violation, for instance, if Alice is a victim of blackmailing and Bob is a police agent. In that case, even if they encrypt their communications, if the blackmailer spots them talking, Alice's privacy would be broken. More examples have arouse in the past

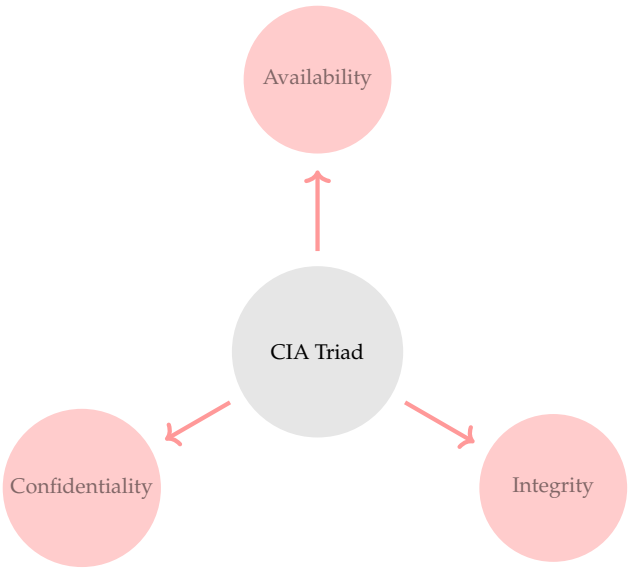


Figure 2.1: The CIA triad.

couple of years, related to massive espionage incidents. Following these guidelines, a (possibly incomplete) definition of privacy may be:

Definition 4 (Privacy)**.** *The quality of a system or protocol ensuring that no private information, nor information that may be processed for inferring otherwise private data, is leaked to other parties than the intended ones.*

One of the main mechanisms for creating privacy respectful systems is anonymity. Anonymity, in turn, may be further classified in subcategories. In this subject, we will use the following related concepts or variations from [163]:

Definition 5 (Anonymity)**.** *The state of being not identifiable within a set of subjects, the anonymity set [163].*

Definition 6 (Pseudonymity)**.** *The use of pseudonyms as identifiers, mainly for authentication [163].*

Pseudonymity is typically used for obtaining *relaxed* types of anonymity. For instance, when the real identity of the participating entities must be preserved, but messages originating from the same entity may be linked between them. This leads us to the unlinkability property:

Definition 7 (Unlinkability)**.** *The property guaranteeing that an adversary cannot do better than random guessing for determining whether or not two or more elements are linked in some predefined way.*

When implementing these different types of anonymity, it is frequent to resort to anonymous communication networks. That is, networks with a special configuration that provide some type of anonymity. Concretely, the most known networks of this type are *mix networks* and those based in *onion routing* (concepts that may be combined too). Generally speaking, these approaches can be defined as follows.

Definition 8 (Mix network)**.** *A* mix network *[63] provides anonymity for the communications routed through it by making each intermediate proxy, known as mix, shuffle randomly all the (encrypted) packets it receives from multiple recipients. By routing the packets through several mixes, tracing them is made harder.*

Definition 9 (Onion routing)**.** Onion routing *[107] is an anonymity enhancing communication model by means of which a packet is routed through several intermediate proxies known as onion routers. The origin encrypts each packet with the public key of each of these onion routers, who then decrypt it upon its reception and forward it to the next router.*

Finally, the use of anonymity may create situations in which this protection is abused in order to perform dishonest or even illegitimate actions[1]. This has led to the creation of systems preventing this misuse, or providing **fairness**, which, following the guidelines in [132], may be defined as follows.

Definition 10 (Fairness)**.** *A system is said to be fair if it guarantees that dishonest or illegitimate actions would be somehow penalized or prevented.*

This property acquires importance in systems where the trusted authorities have too much power (which occurs frequently), where fairness mechanisms may be employed to prevent abuse of power; and in anonymous systems, since its users may take advantage of this anonymity in order to act dishonestly. Fairness may be actually related to *accountability*, which ensures that the entity responsible for some action would be unavoidably held liable for it.

Certainly, there are other important principles that appear less frequently in this thesis (or do not appear at all), like availability (of a resource or of some information), uniqueness of information or freshness, but are also important in certain scenarios.

2.2 Notation

General notation. We use two sets of notation throughout the thesis. One for listings, containing pseudocode and code snippets intended to facilitate the understanding of the code and formal models referenced during this thesis; and another for more abstract descriptions of processes, focusing on the cryptographic details and leaving aside the "programmer point of view". This notation, for general primitives such as encryption and decryption, is summarized in Table 2.1. The notation used for more advanced primitives, such as group and blind signatures, is introduced in Section 2.4 for the general setting, and in Appendix B for the specific abstraction used for the automated security analysis introduced in following sections.

Notation for cryptographic protocols. Specially in Chapter 7, we make use of advanced cryptographic protocols, such as group signatures or partially blind signatures. These are schemes composed by protocols involving complex computations, many times requiring the interaction of several entities. For describing these interactions, and some of the most common operations executed within them, we use the following specific notation.

For a set S, we let $x \leftarrow S$ denote choosing x uniformly at random from S. For an algorithm A, we let $A(x_1, \ldots; r)$ denote the output of A on inputs $x_1, \ldots$ and random

[1] We discuss this subject further in Chapter 6.

Program notation	Abstract notation	Description
`senc`(m,k)	$enc_k(m)$	Symmetric encryption of m (resp. m) with k (resp. k).
`sdec`(c,k)	$dec_k(c)$	Symmetric decryption of c (resp. c) with k (resp. k).
`aenc`(c,k)	$enc_{pk}(m)$	Asymmetric encryption of m (resp. m) with k (resp. pk).
`adec`(c,k)	$dec_{sk}(c)$	Asymmetric decryption of c (resp. c) with k (resp. sk).
`sign`(m,k)	$sign_{sk}(m)$	Signature of m (resp. m) with k (resp. sk).
`verify`(s,k)	$verify_{pk}(s)$	Verification of signature s (resp. s) with k (resp. pk).

Table 2.1: General notation. The column "Program notation" shows the notation used for pseudocode and code listings. The column "Abstract notation" shows the notation used for abstract cryptographic descriptions. Note that for programs we use `teletype` font, while for abstract definitions, we use *math* font. The column Description gives a brief description of each function.

coins r; in addition, $y \leftarrow A(x_1, \ldots; r)$ means choosing r uniformly at random and setting y to the result of $A(x_1, \ldots; r)$. We use the notation $\langle O_A, O_B \rangle \leftarrow Pro(I_C)[A(I_A), B(I_B)]$ to describe a two-party process Pro between parties A and B, where O_A (resp. O_B) is the output to party A (resp. B), I_C is the common input, and I_A (resp. I_B) is A's (resp. B's) private input; when party B does not have output, or both parties obtain the same one, we sometimes write $O \leftarrow Pro(I_C)[A(I_A), B(I_B)]$. Single-party processes are denoted by $O \leftarrow Pro(I)$, with input I and output O. For instance, a Diffie-Hellman key exchange between entities A and B could be denoted with $k \leftarrow DH(g, p)[A(r_A), B(r_B)]$, where k is obtained by both A and B as $g^{r_A r_B} \pmod{p}$. This notation will be mostly employed for systems with high cryptographic load, mostly in Chapter 7 and Chapter 8.

Notation for sequence diagrams. Finally, throughout the chapters of this thesis we include sequence diagrams for depicting the different high level communications protocols that are either presented here or used as examples. In this sequence diagrams, we use the notation in Figure 2.2 for representing the different types of communication channels.

Note that the channels depicted in Figure 2.2 are also accompanied with a literal notation. This notation will also be used throughout the following chapters when giving written descriptions on the channels employed for our protocols and systems. Specifically:

Definition 11 (Public channel (`P`))**.** *A public channel, denoted with* `P`*, is an unauthenticated (for both ends) and unencrypted channel.*

Definition 12 (Confidential channel (`C`))**.** *A confidential channel, denoted with* `C`*, an encrypted channel (we do not differentiate between the possible types of cipher).*

Definition 13 (Origin/sender authenticated channel ($\mathtt{A}_o$))**.** *A channel in which the origin (or sender) authenticate herself by some mean.*

Definition 14 (Receiver authenticated channel ($\mathtt{A}_r$))**.** *A channel in which the receiver authenticates herself by some mean.*

Sometimes, we also use the term *server authenticated channel* when, independently on who starts the communication (thus, on who is the origin/sender and who is the

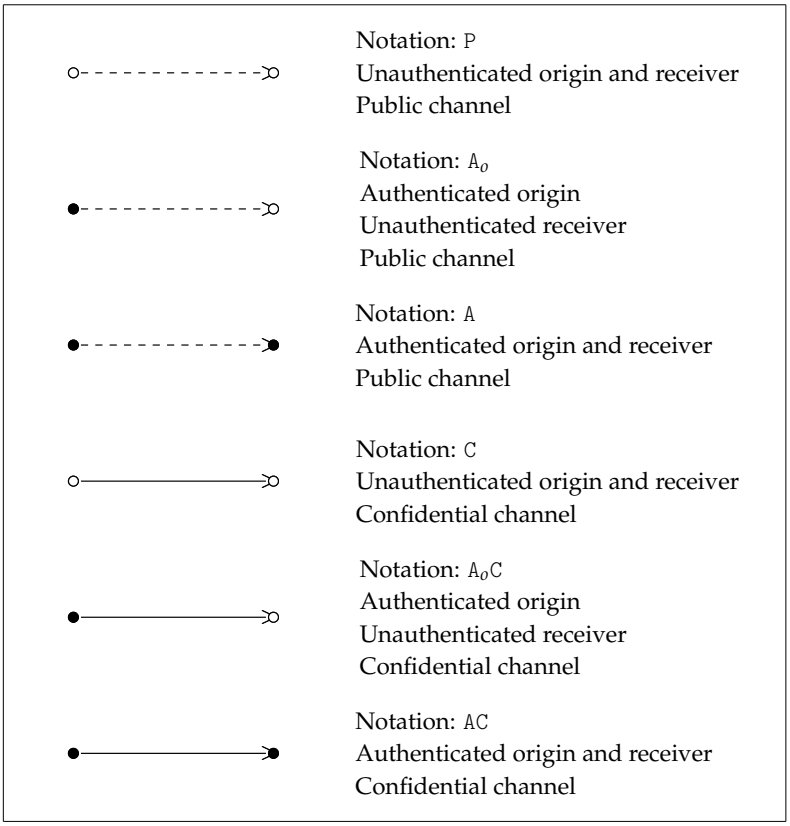


Figure 2.2: Basic notation used for sequence diagrams.

receiver), one of the communicating entities is acting as a server. This is specially relevant in practice, since it is the most frequent configuration for SSL/TLS [89] sessions. Therefore:

Definition 15 (Server authenticated channel (A_S))**.** *In client-server communications, a channel in which the server authenticates herself.*

All these channels address just one property at a time (authenticity or confidentiality). However, it is also possible for them to provide more than one property. In these cases, we combine the notations in a natural manner. For instance, a mutually authenticated and confidential channel is denoted with AC, and a server authenticated and confidential channel is denoted with A_SC(the same combinations apply to the representations in Figure 2.2).

This notation will be mostly used in Chapter 3 and Chapter 5, where we analyze and design communication protocols.

Finally, we many times state that, for our systems, the communications are assumed to be routed through some anonymizing network. That is, we assume that the employed channels provide anonymity. Note that this may be achieved by employing the previously defined *mix networks* or *onion routing* networks, such as Tor [91].

2.3 Security verification

Verifying that the required security properties are achieved is crucial when designing protocols and systems. Therefore, suitable techniques must be applied as a core com-

ponent of security-by-design methodologies, which are a key component in this thesis. Focusing at the design level, there are two main subdivisions, which we call here the *computational approach* and the *formal approach*, following the terminology used in [6].

Security models. First, and independently on the chosen approach, adopting an appropriate security model is a crucial task. A main component of the security model is the *attacker model*, which specifically defines what the attacker can and cannot do in the context under analysis. Consequently, it has direct influence when it comes to prove if the final system complies with those security requirements. This model defines the attacker capabilities that are necessary to conduct the *threat analysis* [15, Chapter 11]. Depending on what can the attacker do, a designer/developer may need to protect different resources or take one approach or another. There exist several classifications:

- An attacker can be said to be *internal* or *external*, depending on whether it is one of the entities which take part in the protocol/system, or a third party that is not included in it.
- An attacker can also be categorized as *passive* if the related attacks consist in observing the messages and a subsequent information inference, or as *active* if he/she actually inserts and/or modifies information on any communication link.
- There are *local attackers* only threatening a subset of an information/communication system elements, versus *global attackers* that can access every component of a system [75].
- Major threats are determined by the so-called *Byzantine attacker*, commonly used as reference when designing fault tolerant systems, and in the context of anonymous communication systems [1, p. 78]. This model considers internal attackers behaving randomly in order to corrupt the system output [136].

The attacker model can be further refined and aligned with the formal and computational verification families by considering the attacker computing capabilities. The *Dolev-Yao attacker* [93], has proven to be a very powerful abstraction when used in conjunction with methodologies for the formal verification of protocols. It assumes an *omniscient* attacker who monitors and can modify the messages sent through all communications channels, but cannot break cryptographic primitives. That is, a Dolev-Yao attacker can insert new messages into any channel, block messages sent by any entity, etc., but she cannot decrypt or sign messages unless she learns the appropriate key. In contrast with this attacker model, there is the *computational attacker* [28], which assumes that attackers are Probabilistic Polynomial Time Turing Machines (sometimes written PPTM or just PPT) having their computing capabilities consequently determined. This implies that the cryptographic primitives are not assumed to be perfect and thus any *breakable* primitive in a PPTM scenario is vulnerable.

Computational approach. In computational verifications, information tokens are treated as bitstrings. Both functions and security are defined using these strings, being security usually probabilistic in terms of the computational complexity needed to break the underlying algorithms. Thus, a deep knowledge of the algorithm details, formal security definitions, cryptographic reduction techniques and complexity theory, is required, as well as specific models compatible with this new context. Typical models are

the *standard*, *random oracle* or *ideal cipher* models; usual methods for defining security are the *game-based* definitions [175], or the *Universal Composability* framework [60] ; and frequent reduction techniques or methods to proof security are the *hybrid argument* [106, Chapter 3, Section 2.3], *game hopping* [175], the *constructive cryptography* approach [143] and of course, *information-theoretic* proofs [144]. An example of this type of analysis is given in [22].

For instance, probably one of the most basic examples may be the Diffie-Hellman (DH) protocol, used for negotiating symmetric keys [90], which runs as depicted in Figure 2.3. In this protocol, all operations are done under a cyclic group G of prime order p, generated by g. In Section 2.3.1, we will see how to model this same protocol using the formal approach.

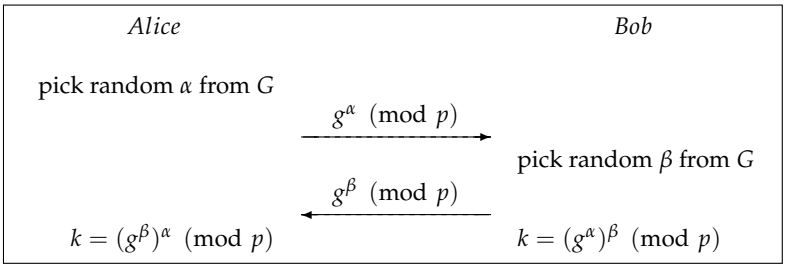


Figure 2.3: Basic DH in which Alice and Bob derive a symmetric key k. G is a cyclic group of prime order p, generated by g.

In a typical computational verification, security of this protocol (in this case, confidentiality of k) would be directly proved based on the Computational Diffie-Hellman (CDH) assumption, which states that no PPTM attacker is able to compute g^{xy} from g, g^x and g^y when using a cyclic group as described above (plus some additional properties to make it secure). Given the CDH assumption, an attacker that is able to derive k is also an attacker to the CDH assumption, which is believed to be secure. For instance, more evolved versions of this protocol including password-based authentication for the Alice and Bob are described in [29], and verified by means of game-based proofs under the ideal cipher and random oracle models in [49].

Formal approach. This approach for security verification uses symbolic logic to create formal models of systems and protocols. Cryptographic functions like encryption or hashing are thus treated as functions over symbolic expressions, abstracting out the inner details of the underlying algorithms. For instance, encryption is assumed to be perfect (and an attacker cannot obtain the plaintext from the ciphertext, unless she also gains knowledge of the corresponding key) and so on: this is again the *Dolev-Yao model* [93] described above. An attractive point of this model is the fact that it is easily adaptable to automated tools that allow for reasonably easy (but powerful) verifications. Such tools are able to consider thousands of alternatives in a few seconds and even provide *soundness* on their conclusions. Some of these formal tools are [17, 26, 39, 41, 74, 159]. Some of the important theories and languages in which these tools are based are the *Spi Calculus* [4], *BAN logic* [51], *equational theory* [118] or *inductive definitions* [160]. An specific example and more insight into this approach is given in Section 2.3.1.

Some examples of important theories and techniques of each approach are depicted in Figure 2.4.

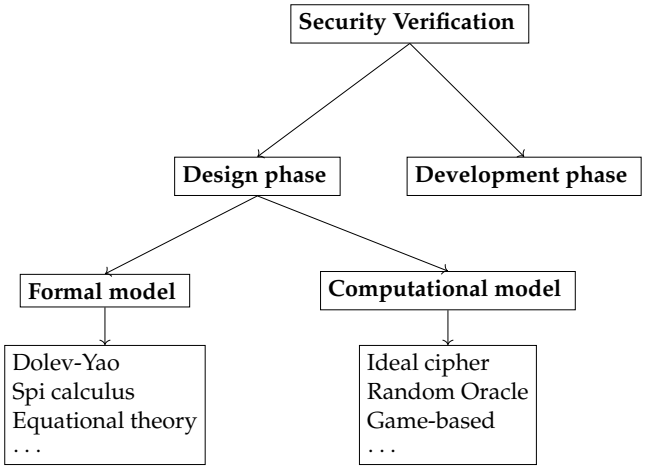


Figure 2.4: Branches of security verification theories and techniques.

2.3.1 Automated verification with ProVerif

During this work, we mostly use the tool *ProVerif*[2] [39] to verify the security properties of several protocols and systems. The reason behind adopting Proverif is that it provides a computer-based (and thus automatic) approach for the verification of security properties. Furthermore, the use of appropriate abstraction models of the protocols being verified allows to apply Proverif to analyze the information flow in order to achieve acceptable security guarantees. However, we emphasize that there exist many other theories and tools for security verification and, many times, rather than choosing one among them, a combination of several would be necessary in order to obtain maximum security.

ProVerif is an automated tool for verifying security properties of communication protocols. More precisely, it verifies these properties over symbolic models of protocols written in a variant of the applied pi calculus [3, 148, 149]. Upon receiving a model of the protocol under perusal in this variation of applied pi calculus, a set of rules (named constructors and destructors) abstracting the equational theory used to compose protocol, and a set of queries, ProVerif applies a resolution process until a point is reached where no more clauses may be added nor removed. Internally, the received clauses are converted into Horn Clauses [117]. Then, from these clauses, ProVerif tries to derive the statements in the previously defined queries.

Among the main properties of ProVerif, stands out the fact that it provides *soundness* and that it supports an *unbounded number of protocol sessions*. The former implies certainty when ProVerif says that a specific security property is satisfied for a given model. Supporting an unbounded number of protocol sessions means in practice that, for instance, ProVerif may simulate the fact that the attacker launches (or forces to launch) several copies of a given process within the protocol, concurrently. This is important since, in that manner, not only sequential executions are taken into account, which is precisely the case in the real world. However, ProVerif does not ensure *completeness*, meaning that it may sometimes be unable to prove properties that hold.

With the help of constructors and destructors, equational theories may be defined as abstractions of cryptographic operations. For instance, Equation 2.1 allows ProVerif to reason about symmetric encryption and decryption. Specifically, it means that using

[2]Specifically, version 1.86.

the same key k in a symmetric decryption function *sdec* over the result of symmetrically encrypting with *senc* the message m with the same key k, yields the original message m. Or Equation 2.2, which captures the core of the Diffie-Hellman problem, stating that $(g^x)^y$ (left-hand side of the equation) equals $(g^y)^x$ (right-hand side of the equation). Both examples are depicted with block diagrams in Figure 2.5.

$$sdec(senc(m,k),k) \rightarrow m \tag{2.1}$$

$$exp(exp(g,x),y) \rightarrow exp(exp(g,y),x) \tag{2.2}$$

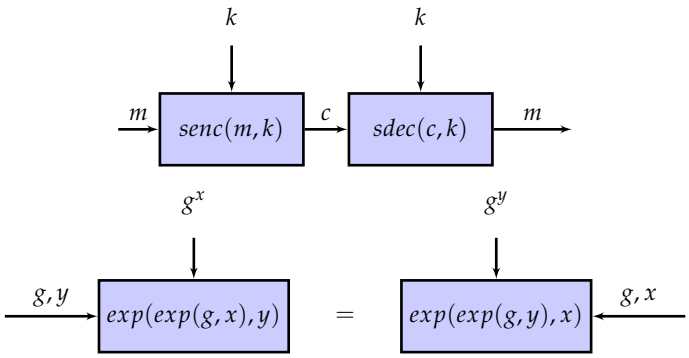


Figure 2.5: Block diagram representation of symmetric encryption and decryption equations, and the Diffie-Hellman equations in ProVerif.

Basic protocol modeling with ProVerif. The relevant parts of a protocol may thus be represented using constructor and destructor rules. The submission and reception of messages between the different entities of the system is represented with the primitives *out* and *in*. These primitives receive as first argument the channel to be used for the communication and, as second argument, the message to be transmitted which may be, for instance, a variable, a name (constant), or the result of applying a constructor.

Additionally, in order to verify specific properties, ProVerif must be queried about them. This implies that ProVerif applies its reasoning model to find attack traces given the defined rule set. Among the security properties ProVerif may be asked about, we can find *secrecy queries* for proving confidentiality, taking the shape `attacker(x)`, which may be read as "*Can the attacker derive* `x`?". *Correspondence assertion queries* for proving authenticity properties, which are basically rules like `eventB(···)` → `eventA(···)`, meaning that each time `eventB` is executed, then `eventA` must have been executed before. By definition, events cannot be executed by the attacker in ProVerif. Therefore, if they occur, it must have been a legitimate process who has invoked it, which makes this a suitable technique for proving authenticity. Finally, ProVerif can also be asked about *observational equivalence*, which serves to prove indistinguishability properties. This is done by including within the code instructions such as `out(network, [term1, term2])`, which roughly asks if the attacker can differentiate a protocol run in which `term1` is sent out via the `network` channel from one in which `term2` is sent instead. Table 2.2 summarizes the main keywords and "constructions" used for protocol modeling in ProVerif, giving informal descriptions of their meaning.

For instance, Figure 2.6 models the Diffie-Hellman key exchange, as depicted in Figure 2.3, using ProVerif typed variation of pi-calculus. Note that we make use of the

Instruction	Informal description
`new t:T`	Create a new term `t` of type `T`
`A|B`	Run processes `A` and `B` in parallel
`fun f(I1,I2):O`	Define constructor function `f` receiving terms of type `I1` and `I2` and producing a term of type `O`
`reduc forall t1:I1, t2:I2; d(f(t1,t2),t2):t1`	Define destructor `d` for recovering `t1` as processed by constructor `f` with `t1` and `t2`
`out(c,m)`	Send message `m` through channel `c`
`in(c,m)`	Receive message `m` through channel `c`
`let t = f(a,b) in`	In what follows, process t as if it was produced by running `f` with parameters `a` and `b`
`attacker(x)`	Secrecy query: Can the attacker obtain `x`?
`eventX(a,b)`	Run `eventX`, which depends on terms `a` and `b`
`eventX(a,b)` $\rightarrow$ `eventY(a)`	Authenticity query: Is `eventY(a)` always preceeded by `eventX(a,b)`?
`out(network, [term1, term2])`	Observational equivalence query: Can the attacker distinguish `term1` from `term2`?

Table 2.2: Main keywords and operations for protocol modeling with ProVerif.

symmetric encryption operation as described in (2.1) and the exponentiation equations as defined in (2.2). The `convert_G_to_key` function is just a special type of function allowing to convert terms of custom type `G` into terms of custom type `key`.

In line 2 of Figure 2.6, we ask ProVerif to try to obtain (i.e., derive from the defined set of rules) newly created `msg` terms. That is, we are asking whether or not secrecy is kept for `msg`. However, since this is an unauthenticated version of Diffie-Hellman, the active Dolev-Yao attacker assumed by default by ProVerif is able to impersonate the receiver `processB`. In fact, the output produced by ProVerif, shown in Figure 2.7, indicates that an attack has been found (note the `A trace has been found` message in line 33). Specifically, it states that `RESULT not attacker(msg) is false`. This means that the attacker does learn `msg` (the token queried about in line 2 of Figure 2.6). The trace described by ProVerif basically states that the attacker receives g^a, generated by `processA`, and obtains g^{ay} for some random y (denoted with `y_110` in Figure 2.7) created by the attacker herself. Then, the attacker sends g^y, which is received by `processA`, who produces g^{ya}. Thus, when finally `msg` is sent encrypted under g^{ya}, the attacker can just decrypt it and obtain the plaintext. This is basically the well-known *man in the middle attack* to *unauthenticated Diffie-Hellman*.

ProVerif is distributed as a free tool[3], and besides several publications in the literature showing the theory behind it [37–39], a comprehensive user manual is also available [42]. Throughout this thesis, we have used the version 1.86 of the software.

[3] `http://proverif.inria.fr`.

```
1  (* Queries *)
2  query attacker(new msg).
3
4  let processA =
5          new a: exponent;
6          out(net, exp(g,a));
7          in(net, gb: G);
8          let k = convert_G_to_key(exp(gb,a)) in
9          new msg:bitstring;
10         out(net, senc(msg, k)).
11
12 let processB =
13         new b: exponent;
14         in(net, ga: G);
15         out(net, exp(g,b));
16         in(net, c: bitstring).
17
18 process
19         processA | processB
```

Figure 2.6: Diffie-Hellman key negotiation modeled with ProVerif. Types `G`, `exponent` and `key` are used to define group elements, exponents and symmetric keys, respectively.

2.3.2 Joint verification with formal and computational appÂąroaches

The previous examples based on the Diffie-Hellman key negotiation, while quite simple, reflect the benefits that may be obtained by applying both the computational and formal verification approaches. In the computational case, the protocol was evaluated as secure based on the CDH assumption. However, with the formal approach, we saw that it is trivial for an active attacker to impersonate one of the parties, thus breaking confidentiality. Conversely, a formal verification may dictate that a protocol is secure, but performing a computational verification may find design flaws because, e.g., an incorrect complexity assumption has been made.

That is, while the computational model is useful to guarantee the security of the underlying primitives, the formal model becomes fundamental for ensuring that there is no possible attack based on arbitrary compositions of the protocol operations. For instance, applying the formal model is easy to check that the good practices endorsed in [5, 16] are kept, as well as many other advanced features, assuming the Dolev-Yao model [93] and including complex combinations of message replays, impersonations, etc. Moreover, it can be done with the help of automated tools. Once this has been analyzed, we can apply computational verification techniques in order to prove that the Dolev-Yao requirements are actually met. Additionally, it is also of independent interest the efforts to bridge both approaches [6, 126].

Considering this, a clear context in which we can take advantage of the features of both approaches is in the design of communication protocols with strong dependence on cryptographic systems, which is the case of this work.

```
1. We assume as hypothesis that
attacker(y_110).

2. The message exp(g,a[]) may be sent to the attacker at output {2}.
attacker(exp(g,a[])).

3. By 2, the attacker may know exp(g,a[]).
By 1, the attacker may know y_110.
Using the function exp the attacker may obtain exp(exp(g,y_110),a[]).
attacker(exp(exp(g,y_110),a[])).

4. Using the function g the attacker may obtain g.
attacker(g).

5. By 4, the attacker may know g.
By 1, the attacker may know y_110.
Using the function exp the attacker may obtain exp(g,y_110).
attacker(exp(g,y_110)).

6. The message exp(g,y_110) that the attacker may have by 5 may be
received at input {3}.
So the message enc(msg_115,exp(exp(g,y_110),a[])) may be sent to the
attacker at output {6}.
attacker(enc(msg_115,exp(exp(g,y_110),a[]))).

7. By 6, the attacker may know enc(msg_115,exp(exp(g,y_110),a[])).
By 3, the attacker may know exp(exp(g,y_110),a[]).
Using the function dec the attacker may obtain msg_115.
attacker(msg_115).

[..]

A trace has been found.
RESULT not attacker(msg[gb = v_44]) is false.
```

Figure 2.7: Attack trace found by ProVerif for unauthenticated Diffie-Hellman exchange.

2.4 Cryptographic primitives

Next, we give a high level overview of the main cryptographic primitives that we have resorted to for implementing privacy in the the protocols and systems described Part III and Part IV of this thesis. Probably, the most omnipresent one are group signatures, for which we also provide an open source library in Chapter 4. However, we also make use of other schemes, like partially blind signatures, zero-knowledge proofs, and commitments.

2.4.1 Group signatures

As stated, *group signatures* are a cryptographic primitive that takes a central role for ensuring privacy in the systems we propose. Group signatures, first described in [65], are like conventional digital signatures in that they are used to prove that the owner of a specific secret has been the source of some information. However, unlike their conventional counterpart, group signatures hide this owner among a set (group) of

possible owners, hence providing anonymity.

Around this central property, several variations of group signatures have been proposed, enabling additional features. For instance, typically, there is a *Group Manager*, who controls some secret information that allows him to revoke this anonymity, and fetch the identity of the issuer of a group signature (this is called *opening* a group signature). But there also exist schemes, like *ring signatures* [170], that provide unconditional anonymity, meaning that the open functionality cannot be performed. Other schemes, like the ones proposed in [67, 132], add an extra trapdoor besides the one used for opening group signatures so that an authority (either the Group Manager or some subsidiary authority) is able to link signatures made by the same group member, but instead of using his/her identity, a *tracing trapdoor* used solely for this task. This is called *tracing*, and this type of signatures are consequently named *traceable signatures*. [132] also adds the functionality to claim, in zero-knowledge, having issued a specific group signature. Even though the term is not used in the original paper, we could name variations supporting this functionality *claimable group signatures*. In [32], the trust placed in the Group Manager is divided across several authorities, which need to combine their secrets in order to be able to open some group signature. The authors call the result *fair signatures* or, rather, fair traceable signatures, since their proposal is based on traceable signatures (and also supports tracing). Besides these extensions to their functionality, their efficiency has also been refined. A detailed overview of the evolution of the computational and communication costs of the different schemes of group signatures is available at [137, Sec. 1.1].

According to the previous introduction to group signatures, for different schemes the different operations may imply subtle differences, e.g., in the way they are performed or their implications with respect to the privacy of the group members; or they may just implement a subset of the mentioned functionality. However, it is possible to abstract the functionality from the study of the main primitives. In Figure 2.8 we sketch an abstraction of all the operations provided by group signatures, mostly matching the ones described in [132] that we will use hereafter. Each operation is described as follows:

Setup. Generates and initializes the group and manager keys for any arbitrary group. All operations below are always related to a group initialized with this operation. This operation is denoted with:

> $(pk_G, sk_G) \leftarrow \texttt{GS.Setup}(1^k)$. Creates group public key pk_G and private Group Manager key sk_G for group G.

Join. The process by means of which new members join the group. It is typically divided in a phase run by the new member, who obtains a member key, and a phase run by the group manager, who updates the Group Membership List (GML). This operation is denoted with:

> $\langle mk_i, \ell' \rangle \leftarrow \texttt{GS.Join}(pk_G)[M(s_i), GM(\ell, sk_G)]$. Allows member M providing secret s_i to join group G, generating the private member key mk_i and updating the Group Membership List ℓ to ℓ'.

Sign. The process of issuing a group signature. This operation is denoted with:

> $\varrho \leftarrow \texttt{GS.Sign}_{mk_i}(msg, mk_i)$. Issues a group signature ϱ on msg using member key mk_i.

Verify. The process for verifying a group signature. This operation is denoted with:

> $\texttt{GS.Verify}_{pk_G}(\varrho, msg)$. Verifies whether ϱ is a valid group signature on msg and group G.

Claim. The process for claiming ownership of a group signature. This operation is denoted with:

> $\pi \leftarrow \texttt{GS.Claim}_{mk_i}(\varrho)$. Allows a member of group G to create a (zero-knowledge) claim π of the ownership of ϱ.

- **Equality proving**. The process by means of which the issuer of a set of group signatures proves that she has issued all the signatures within the set. This may be seen as a generalization of the claiming process, and is specially interesting in schemes allowing zero-knowledge claims, like [67, 132]. The notation for this generalization is shown in Section 2.4.3.

Claim Verify. The process of verifying a claim of a group signature. This operation is denoted with:

> $\texttt{GS.ClaimVerify}_{pk_G}(\pi, \varrho)$. Verifies if π is a valid claim over ϱ.

- **Verification of equality**. The verification counterpart for equality proving. The notation for this generalization is shown in Section 2.4.3.

Open. Used for extracting the identity of the issuer of a specific group signature. This operation is denoted with:

> $i \leftarrow \texttt{GS.Open}_{sk_G}(\varrho)$. Opens group signature ϱ, returning the identity i of the member of G who issued it.

Reveal. Employed for extracting the tracing trapdoor of a group member.

> $t \leftarrow \texttt{GS.Reveal}_{sk_G}(\varrho, \ell, r)$. Using the Group Membership List ℓ and the group manager key sk_G, produces the tracing trapdoor t associated with the issuer of ϱ. If required, t may be added to a revocation list r.

Trace. Checks whether a group signature has been issued by a group member who has been somehow revoked, e.g., included in a Certificate Revocation List (CRL).

> $0/1 \leftarrow \texttt{GS.Trace}(\varrho, r)$. Using the Revocation List r outputs whether the member key of the issuer of ϱ has been revoked (1) or not (0).

2.4.2 Partially blind signatures

Blind signatures where introduced by Chaum in [64]. Basically, a blind signature scheme allows a user U to obtain a signature from a signer S over any arbitrary message m, but without S learning anything about m. This idea has been used since then for creating privacy respectful systems, like e-cash based e-commerce. However, since the signer does not learn any information about the message, systems based on them can easily be abused. For solving this issue, *fair blind signatures* were proposed in [176]. In this variant, an authority has privileged information allowing the signer to link message

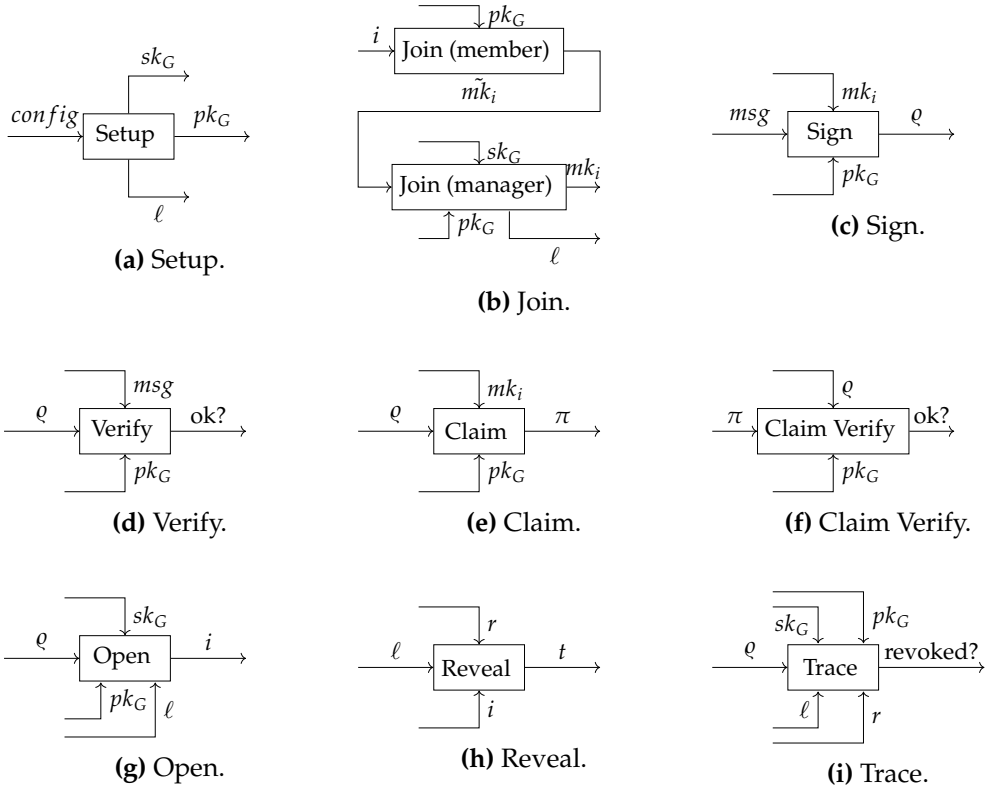


Figure 2.8: Main operations in group signatures. Incoming arrows depict inputs and outgoing arrows depict outputs.

and signature pairs. *Restrictive blind signatures* [47] allow issuing blind signatures, but only choosing among messages that comply certain rules. Finally, an also important alternative is given by partially blind signatures [7]. In a *partially blind signature* the messages are divided in two parts: a common message to which the S has complete access; and the blinded message, of which S does not learn anything. Thus, the common message may be employed to implement misuse prevention mechanisms. As always, several schemes have appeared improving the overall efficiency, reducing the size of the final signatures, or based on different number theory problems [43, 66, 129, 156].

In general, a partially blind signature scheme supports the following operations (note that a blind signature may be seen just a special case of a partially blind signature, where no common message is sent):

Setup. Creates the signer's public and private keys. This is denoted with:

> $(pk_S, sk_S) \leftarrow \texttt{PBS.KeyGen}(1^k)$. Creates a public key pk_S and a private key sk_S for issuing partially blind signatures.

Blind. The user creates a blinded version of the message to be signed (the blinded message). The public key of the signer is required for this action. This is denoted with:

> $(\tilde{m}, \pi) \leftarrow \texttt{PBS.Blind}(m, r)$. Run by a user U, it blinds the message m using a secret value r. It produces the blinded message $\tilde{m}$ and a correctness proof

π of the produced blinded message.

Sign. Upon receiving the blinded and common messages, the signer runs any necessary verification and creates a blinded signature using its private key. This is denoted with:

$\tilde{\varrho} \leftarrow \texttt{PBS.Sign}_{sk_S}(cm, \tilde{m}, \pi)$. Signer S verifies proof π and issues a partially blind signature $\tilde{\varrho}$ on $(cm, \tilde{m})$, where cm is the common message.

Unblind. The user receives the blinded signature and unblinds it, probably using some random secret value generated during the blind process. The result of this operation is the final signature. This is denoted with:

$\varrho \leftarrow \texttt{PBS.Unblind}_{pk_S}(\tilde{\varrho}, \tilde{m}, r)$. Run by the user U, who verifies $\tilde{\varrho}$ and then uses the secret value r to produce a final partially blind signature ϱ.

Verify. Any entity runs this operation to verify the signature. For this, it is necessary the common and blinded messages, and the public key of the signer. This is denoted with:

$\texttt{PBS.Verify}_{pk_S}(\varrho, cm, m)$. Verifies if ϱ is a valid partially blind signature on (cm, m).

2.4.3 Additional primitives

There are also other cryptographic primitives that are typically used as the core of cryptographic protocols and systems for the fair management of privacy. Specifically, we make use of commitments and zero-knowledge proofs.

Definition 16 (Commitment)**.** Commitments *[48] allow entities to commit to some value such that they may operate with it without revealing it until it is required, but being able to prove that they have not modified the value (i.e., they have been committed to it).*

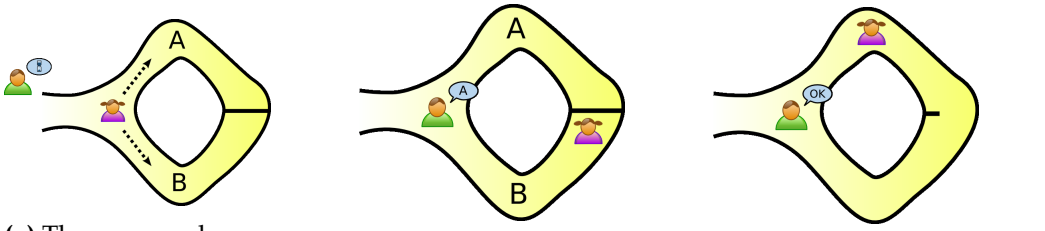
For instance, given a cyclic multiplicative group of order p, with generator g, sending $c = g^x$ is a commitment to x and subsequently revealing x proves the commitment (revealing x is called decommit). If hiding any information about x that may be learned from the commitment is necessary, then $c = g^x h^r$ may be used instead, where h is another generator of the group, and r is a random value. This is an example of a Pedersen commitment [161]. Commitments will be denoted with $\mathsf{com}_m \leftarrow \mathsf{Com}(m; r_m)$, where com_m denotes a commitment to a message m in which the sender uses uniform random coins r_m; the sender can open the commitment by sending (m, r_m) to the receiver.

Definition 17 (Zero-knowledge proofs and Zero-knowledge proofs of knowledge)**.** Zero-knowledge proofs *[108] allow an entity A to prove to another entity B that certain statement is true, but without revealing any other information than the truth of the statement. Of particular interest for privacy systems are* zero-knowledge proofs of knowledge*, i.e., A proving to B that she knows something, but without revealing it (for instance, proving knowledge of a password).*

Figure 2.9, from Wikipedia [4], depicts the concept of a zero-knowledge proof as explained in [166]. Alice wants to prove to Bob that she knows the password for opening

[4] `http://en.wikipedia.org/wiki/Zero-knowledge_proof`. Image uploaded by user *Dake*. Last access, March 29th, 2015.

the secret passage connecting the tunnels in a cave. Alice enters in the cave, randomly choosing which branch to follow. Bob then shouts which exit he wants her to take, and sees whether she is able to come out correctly or not. After Alice succeeds in n attempts (with certainty increasing with n), Bob will be convinced that Alice knows the password, but without learning it.



(a) The prover chooses randomly the input path.

(b) The verifier chooses randomly the exit path.

(c) The prover returns. After repeating the experiment n times, the verifier will be convinced that the prover knows the password.

Figure 2.9: Pedagogical example of zero-knowledge proofs. Source: Wikipedia[4].

We use $\pi \leftarrow \texttt{ShowZK}(x; w)$ and $\texttt{VerifyZK}(x, \pi)$ to refer to creating non-interactive proof π showing that the statement x is in the language (which will be determined by the context) with the witness w, and to verifying the statement x based on the proof π. Specifically, as stated in Section 2.4.1, we use zero-knowledge proofs for showing that multiple group signatures have been issued by the same group member. This would be denoted with $\pi \leftarrow \texttt{ShowZK}(\varrho_1, ..., \varrho_n, mk_i)$, and the corresponding verification with $\texttt{VerifyZK}(\varrho_1, ..., \varrho_n, \pi)$.

Part II

Base components

In this second part, we describe two important components that have been employed for designing and implementing the protocols and systems in Parts III and IV. Specifically, Chapter 3 describes a design methodology including the general verification principles described in Section 2.3 within a design lifecycle for protocols and systems, aimed to ease the task of creating secure designs. In Chapter 4, we describe an open source, easily extensible, C library for group signatures that has been implemented with the aim of easing the use of this powerful primitive in privacy respectul systems. As such, it has been integrated as a core component in the prototypes implemented for evaluating the efficiency of the proposals in Parts III and IV.

CHAPTER 3

A methodology for designing secure protocols and systems

Chapter based on and supported by references [22, 82, 83].

Security vulnerabilities have a severe impact on computer and communications systems and the companies maintaining them. In our society, many critical components (some known as *critical infrastructures* after our dependence on the services they provide) depend on information and communications security, and are put at risk if their components are not properly secured: energy infrastructures, health care systems, military communications, banking transactions, etc. For users, and independently on what security property is weakened or broken (e.g. confidentiality, authenticity, integrity or availability), the result many times reduces to a threat to their privacy. But, as pointed out in [8], privacy compromises may also yield important economic losses for the companies (or data holders, as called in [8]).

In [105], the authors analyze the financial effects of software vulnerabilities, and propose an alternative methodology to the Common Vulnerability Scoring System (CVSS[1]) in order to take this factor into account when evaluating the urgency to solve bugs. They produce an estimation of the average response costs, depending on the severity class of the vulnerability. Their estimations are partially reproduced in Table 3.1.

Severity class	Average response costs (EUR)
Low	700
Medium	1.5K
High	3K
Critical	7.5K

Table 3.1: Average response costs. From [105].

The above mentioned costs mostly correspond to implementation problems. These issues are almost always solvable, despite the additional costs. On the other hand, design-level errors, which are the focus of this chapter, are usually harder to address. Specifically, a design flaw requires modifications in the system's design, which may not be applicable if the system is already deployed. Thus, a design error leaves just two bad choices: either do not fix it and assume the related security risks; or fix it and

[1] `http://www.first.org/cvss/cvss-guide.html`. Last access on March 31st, 2015.

probably assume enormous costs (in economic and re-deployment terms). Examples of important design errors are present in the WEP Shared Key Authentication protocol [119], which allowed passive attackers to circumvent authentication in wireless networks (more on this in Section 3.3.2) and was one of many factors leading to a deprecation of WEP; the Bitcoin malleability issue[2] which allowed flawed implementations to accept transactions that could later be invalidated despite being fundamentally correct, e.g. allegedly causing the closure of Mt. Gox[3], one of the most important Bitcoin exchangers; or the Email Based Identification and Authentication (EBIA) [102] which ignores the fact that conventional email messages use insecure communication channels and thus are not suitable, by themselves, to guarantee a secure authentication.

Still, even simple design errors are present in many wide-spread systems. This may be due to yet extended ignorance on security-aware design methodologies, or in the unsuitability of the already existing ones. Therefore, applying a comprehensive procedure that helps to avoid design flaws is necessary. Actually, this should always be kept in mind when designing a new system or protocol and is what the security-by-design endorses through the introduction of security verification techniques as a core component of the design process, instead of just an optional step at its end. In this chapter, we present a methodology for analyzing whether the security properties of newly designed protocols and systems are met. The proposed methodology analyzes the given system or protocol in an iterative manner and with increasing complexity as the analysis goes on. If design flaws are found, feedback is obtained for knowing which specific part(s) is (are) involved. Additionally, it gives guidelines for the formalization of the properties important for security, therefore easing any necessary formal verification by means of any existing theory or tool. This is also important, since despite the fact that many tools and theories exist for performing formal security verification and proofs, sometimes it is not clear what properties should be checked at a high level.

[2] `https://en.bitcoin.it/wiki/Transaction_Malleability`. Last access on March 31st, 2015.

[3] `http://en.wikipedia.org/wiki/Mt._Gox#February_2014_shutdown_and_bankruptcy`. Last access on March 31st, 2015.

3.1 Related work

There exists plenty of previous work in the subject of security verification and proofs of security of systems and protocols, and many classifications could be made. In this section, we summarize some of the existing work, at a very high level, starting with the point of the in which this verification can be made and following with the different techniques that may be applied.

Concerning the moment in which verification techniques are applied, there are two main types of frameworks aimed to the task of creating secure systems and protocols: frameworks applied at the *design phase* [2, 115, 130, 142, 174] and frameworks applied at the *development phase* [34, 178]. On this point it is highly relevant the effort in [14] to bridge both frameworks. In that work the authors propose a method to design security protocols, to verify the underlying security properties, and to derive in a automatic way Python and/or C++ implementations of the corresponding systems.

Frameworks applied at the development phase are used, for instance, for checking specific code implemented in concrete programming languages [11, 12, 30, 34, 61, 111, 134, 146]. For this task, there also exist many industry tools, like Coverity[4], HP Fortify[5], IBM AppScan[6], Klockwork[7], Parasoft[8] or Veracode SAST products[9]. Other tools allow verifying runtime errors and test the security of the produced applications by observing their behavior (this is called dynamic analysis) and interacting with them. Some of the previous tools also allow this behavior. Among this last type we may find tools like Veracode's DAST products, vulnerability testing frameworks like Metasploit[10], or the so called fuzzy testing tools, like Codenomicon[11] or Peach[12]. Obviously, all these tools are essential for creating secure and quality software products. However, they are inescapably linked to the underlying technology.

In the field of design-level verification theories and techniques, a summarized review of them was made at Section 2.3. In this case, one of their disadvantages is that, while obtaining a design that is secure for the desired purpose, implementation errors may still occur. Thus, the combination of both frameworks is necessary. In addition, as mentioned in Section 2.3, sometimes it is hard to translate a set of informal security requirements into formal ones. For instance, taking into account the security risks that may be posed by some external and probably hard to foresee facts is many times ignored.

There are plenty of tools applied to the automatic verification of security properties. In Figure 3.1, we show a summary of the such tools according to our taxonomy (tools focused on the systems design phase versus tools dealing with the development phase).

In the specific area of the verification of protocols design, it is hard to determine the security requirements of each specific component. For instance, looking into the STRIDE methodology [115, 174], it sets general threats (specifically: spoofing, tampering, repudiation, information disclosure, denial of service and elevation of privilege) and associates security properties to each of them (authentication, integrity, non-

[4] `www.coverity.com`. Accessed on December 25th, 2014.
[5] `www.fortify.com`. Accessed on December 25th, 2014.
[6] `www-03.ibm.com/software/products/en/appscan`. Accessed on December 25th, 2014.
[7] `www.klocwork.com`. Accessed on December 25th, 2014.
[8] `www.parasoft.com`. Accessed on December 25th, 2014.
[9] `www.veracode.com`. Accessed on December 25th, 2014.
[10] `www.metasploit.com`. Accessed on December 25th, 2014.
[11] `www.codenomicon.com`. Accessed on December 25th, 2014.
[12] `www.peachfuzzer.com`. Accessed on December 25th, 2014.

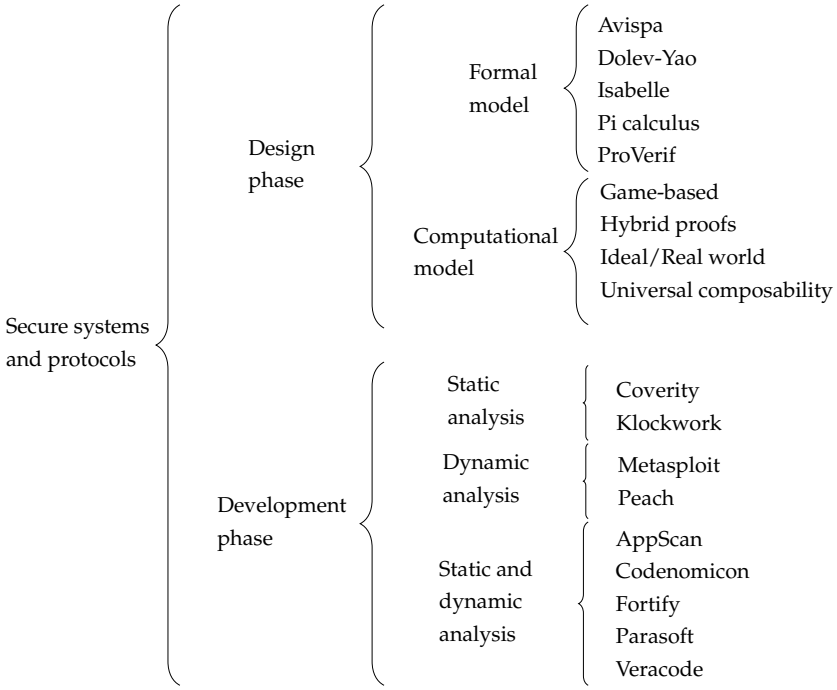


Figure 3.1: Frameworks and tools for secure design and development.

repudiation, confidentiality, availability and authorization). Then, during the design phase, if it is found that a certain resource may be affected by a threat, STRIDE advises to apply specific procedures like encryption or hashing (depending on the threat, of course). However, just applying cryptographic primitives may not guarantee the defined security requirements. Certainly, we have to bear in mind that systems security is an overall properity that is affected by additional aspects as the timing of cryptographic tokens or the procedures to generate and distribute such tokens. On the other hand, methodologies and tools like [14, 130] provide detailed guidelines or means on how to model cryptography into final protocols or systems, and verify the security requirements. Nevertheless, these tools deal with the application or abstractions of cryptographic primitives, which calls for a formal definition of the protocols from the very beginning of the security analysis. Consequently, too much effort is demanded even to detect the more simple flaws.

With the methodology described next, we attempt to ease this task of producing formal security requirements from informal ones, also taking into account elements that may imply threats to the security of the system, while not being part of it. The methodology begins with an initial informal verification approach, that reduces the costs of identifying trivial flaws, while improving the overall knowledge about the protocol or system under analysis. Moreover, it is not intended to substitute any existing framework or tool for the verification of security properties, but to complement it. Indeed, it is aimed at allowing an easier and less costly transition from an informal definition and verification of security systems/protocols to their formal definition and verification.

3.2 A methodology for designing secure protocols

In this section, we describe an adaptive and recursive process for designing secure information protocols. This methodology is built upon evaluation tools for verifying the achievement of the defined goals, based on predefined and detailed attacker and communication channel models. Any threat or flaw identified at any point is handled as feedback to further improve the system design or implementation in subsequent iterations of the methodology. Moreover, the methodology is divided in two main stages of increasing complexity. This twofold methodology enables the detection of simple flaws with less effort, proceeding to more consuming tasks only when (most probably) just complex flaws may be present.

The first stage of the methodology comprises a context and model definition, and an informal verification of an initial design. If this initial analysis is passed, the methodology then proceeds with a procedural analysis[13], which may either follow the formal or the computational approaches. During this work, we mostly make use of the formal approach. However, we emphasize that the computational approach is perfectly compatible with the methodology.

We first give some insight on the main aspects of the methodology, and subsequently proceed to define the steps that compose it. A former version of this methodology was completely described in [79, 82]. In what follows, we describe the main aspects that our methodology takes into account.

3.2.1 Context analysis

Defining what can and cannot attackers do within our systems, the assumptions that we make concerning the entities that take part in it, and the communication channels used to exchange information between them, are essential for producing realistic designs. Otherwise, any conclusion reached through either formal or informal analysis will not successfully adapt to the scenario in which our system will run.

Attacker capabilities. Typically, in security proofs, only general attacker capabilities are taken into account. For instance "the attacker has PPT-like computational power" or "the attacker may use an encryption oracle". However, there are other capabilities that need to be taken into account during the design, in order to create realistic protocols. For example, which information tokens can be easily accessible to an attacker by alternative means, like Facebook pages or public databases, or the fact that an active attacker may easily find the information required to impersonate an arbitrary user during a signup process (e.g., through social engineering techniques). While these details are probably too fine-grained for them to be considered in the abstractions of cryptosystems used for security proofs, the effects that they could have on the security of the final system may be devastating. Yet, they can already be detected and considered at the design stage. Consequently, our proposal includes a finer control on the attacker capabilities during the design of the protocol, and sets explicit control points that provide feedback when security errors are found, easing their correction in subsequent iterations. Also, our proposal supports the application of procedural (either formal or computational) verification methods to reach the highest assurance levels of other methodologies.

[13] We use the term *procedural analysis* to avoid confusion with the formal approach for security verification, although, when it is clear from the context, we may use *formal analysis* instead.

Entities assumptions. In the same way that modeling the attacker capabilities is a necessary task, specifying the expected behavior of each entity taking part in the protocol is also mandatory. That is, it is needed to state whether we expect all entities to behave honestly, semi-honestly or dishonestly. Specifically:

Definition 18 (Honest entity)**.** *A* honest entity *is one that follows the protocol without performing any action that is outside the "rules" specified within it, and does not leak any information to the attacker*

Definition 19 (Semi-honest entity)**.** *A* semi-honest entity*, while follows the protocol and does not leak any information to the attacker either, will try to learn anything it can from the received information.*

For instance, a mail provider considered as a semi-honest entity may be assumed not to share the contents of the emails it manages (and to follow the protocol of sending/receiving emails), but use it for business/marketing intelligence techniques in order to increase its benefits offering targeted advertising.

Definition 20 (Dishonest entity)**.** *A* dishonest entity *tries to subvert the protocol and is assumed to share all its information with the attacker (hence, it may be considered as part of the attacker).*

Communication channels abstraction. Communication channels also play a key role in secure protocols. Therefore, they must be considered during the design stage. For instance, common sense dictates that if you have a piece of information that needs to be authenticated, you either have to send it through a channel guaranteeing authenticity, or you have to use a Message Authentication Code (MAC) or equivalent function to transmit it over an insecure channel. This is a verification simple enough to be performed by hand in many cases, but still can help detect common flaws. Moreover, by getting rid of these simple flaws at the beginning of the design process, subsequent phases may focus on the analysis of more complex issues. In the proposed methodology, this preliminary analysis is done during the first half of the design process.

3.2.2 Verification

As we have discussed in Section 3.1, the formal approach for the verification of cryptographic systems is prone to be included in automated tools. Even though there are many of those tools, in our studies we have mainly applied *ProVerif* (see Section 2.3.1)[14] [39]. Therefore, most of what we explain below concerning the analysis of real protocols is focused on the application of ProVerif. However, we emphasize that any other tool may be equally applied in stead of ProVerif.

Finally, our methodology is compatible with any computational approach verification. Even though we find that automated proofs in the formal model are usually more *engineer-friendly*, and thus may be more suitable for a first procedural analysis, the computational variant certainly increases the security guarantees by further endorsing the obtained results. Moreover, computational analysis complements the formal one with additional considerations that are otherwise not taken into account, like the hardness of executing specific attacks. Specially, those that take advantage of the algebraic properties of the underlying cryptosystems.

[14] `http://proverif.inria.fr/` (accessed April 23rd, 2014)

With all this taken into account, our methodology, when applied in its totality, would provide the assurance level PAL3 (PAL stands for *Protocol Assurance Level*) as defined in [142] (and even the PAL4 level, qualification therein suggested for protocols also verified in the computational model).

With these considerations, we say that a *design flaw* occurs when any security property required to a given protocol under analysis, as stated in the protocol goals, cannot be satisfied given the defined security context derived from the attacker capabilities, communication channels abstraction, entities assumptions and protocol candidate.

3.2.3 Description of the methodology

The proposed methodology is divided in two main phases. The first one initiates an informal definition of the security model, attacker capabilities, communication channels abstraction and protocol, including a manual verification. The second phase formalizes the previous informal definitions, and proceeds with a formal verification. These two phases are executed iteratively, until no flaw is detected. By making a division in the mentioned stages and applying an iterative flow, we allow for easier problems to be detected with less effort and in the early steps, thus saving costs in the process (a simple flaw going unnoticed until an advanced verification stage incurs in higher costs than in the case of being detected earlier, just because the time lost during the advanced verification). The methodology steps are depicted in Figure 3.2, and are defined as follows.

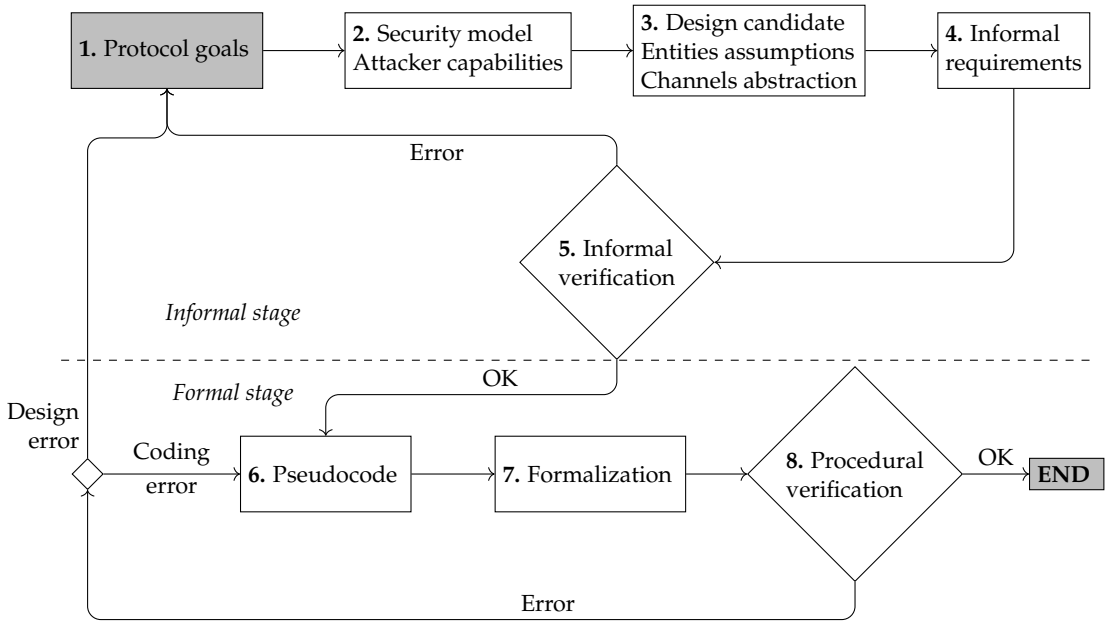


Figure 3.2: Proposed methodology for designing secure protocols. First, it runs an initial informal verification, setting the goals and requirements. Secondly, it formally verifies the security requirements.

Phase I: Analysis of security context and informal verification. This first part is mainly intended to avoid incongruences related to security requirements that are un-

achievable or inappropriate for some reason. It also allows us to perform several informal checks to our first design candidates, and improves our understanding of the protocol. This part is depicted in the first phase of Figure 3.2. Namely, here we face the following matters:

Step 1. Goals of the protocol. Within this step we have to create an informal specification of what we intend to achieve with our protocol, something like *"Complete certainty that the person being registered in our website is who he/she says he/she is"*. This step might seem too basic to be needed, but specifying the security expectations for the protocol will help to improve its understanding and will be useful for the third and fourth steps.

Step 2. Attacker capabilities and security model. This step is essential, because depending on the capabilities the attacker has, some security requirements might or might not be attainable. Moreover, the type of attacker we choose will determine what tools or methods we are going to need in the next steps. First we have to take into account all the different *general capabilities* that our attackers will have. We have to determine whether we are only worried about internal/external attackers, Dolev-Yao/computational models, etc. Once we have established these general capabilities, we have to add or remove any finer level capability that may influence in the system. As a result, with this step we will get a specification of the attacker model, e.g. the Dolev-Yao or computational model, along with a list of specific capabilities added or subtracted to it, like *"Knows the home address of everyone that may be involved in the protocol"* or *"Cannot eavesdrop wired communications"*. Henceforth, we will refer to this list as the *"+/- capabilities"* list.

Finally, note that in case we are not in the first iteration of the methodology, the output of both steps 5 and 8 may be used as input to this step. Indeed, verification failures might cause us to reconsider the attacker model, in case it was not realistic (either too optimistic or too pessimistic).

Step 3. Design candidate, Entities assumptions and channels abstraction. Once we have pointed out the goals of our protocol, the attacker capabilities and security model we adopt, we can cast a design candidate. A sequence diagram or a description using the informally known as *security protocol notation*[15] may be suitable alternatives.

At this point, we also need to consider here what do we expect from the entities that will take part in the protocol. That is, which entities will always behave honestly, semi-honestly or dishonestly. This will affect the security measures that need to be taken in order to prevent attacks on our model.

We also produce here a detailed list of the communication channels that our design candidate makes use of (for instance, following the definitions given in Section 2.2). That is, for each of them, we have to specify the security properties it provides, and how does it provide them. Of course, using a hypothetical perfectly secure channel would trivially give place to a perfectly secure protocol. However, the optimal choice is almost always to place the less restrictive requirements on the employed channels.

[15] `http://en.wikipedia.org/wiki/Security_protocol_notation`. Last access on March 31st, 2015.

Like in step 2, for iterations of the methodology other than the first, we should consider here the informal or formal requirements that failed in the previous iteration, in order to propose a solution to the encountered problems.

Step 4. Informal requirements. This step of the methodology is the first part of an informal verification procedure. Before undertaking the time consuming formal verification task, it is worth making a simpler manual verification in order to detect simple mistakes. For that purpose, we analyze the design candidate thoroughly (but manually) by assigning *informal requirements* to each component of the sequence diagram obtained in the previous step. Note that we use the term *informal requirement* since the evaluation produced at this stage is also informal. The procedure is as follows: for each step of the sequence diagram we note down the *informal requirements* we expect from each of its elements to keep. If within a specific step we apply some function to merge/disassemble several elements, we may have to require additional *informal requirements* (e.g., a message containing the a priori unauthenticated elements x_1, x_2 along with $MAC_K(x_1, x_2)$ can be considered authenticated by the holder of a key K, if the MAC verification succeeds provided that the key K is trusted)[16]. Since the properties we require from a specific element may change through the protocol, we will revise them for every step, even if the element has already been processed previously. This process is summarized in Algorithm 3.3. For instance, we may use as requirements: *Authenticity*, *Confidentiality*, *Integrity*, *Uniqueness*, or *None* (when an item does not require any specific informal requirement).

```
Data: The sequence diagram of the protocol.
Result: A set of informal requirements for each protocol step.
for s = first step; until s = last step do
    for e = first element until e = last element of step s do
        Set reqs[s][e];
    end
    reqs[s] += Additional informal requirements for step s;
end
```

Figure 3.3: Algorithm for assigning requirements to protocol elements and messages. With elements we refer to any component calculated or sent within each specific step.

Therefore, as output of this step we get a list of *informal requirements*. Optionally, this list can be depicted jointly with the sequence diagram to create a requirement-tagged sequence diagram of our protocol design candidate.

Besides, we also have to decide in this step which verification tool or process we will apply in the second phase. Now that we know our specific *informal requirements*, we can choose an appropriated tool (depending on the security properties we want to verify).

Again, if this is not the first iteration of the methodology, we would be required

[16]Typically, this kind of rules are formally specified inside formal verification tools (e.g., Cryptyc [17] is based in type deduction rules). However, since this step is intended to be a preliminary informal approach, common sense and experience are enough.

to reconsider the points in the design where our (informal or formal) model of the protocol failed.

Step 5. Informal verification. Now we have to process the +/- *capabilities list* and security model produced at step 2, the design candidate and communication channels abstraction obtained at step 3 and the *informal requirements* of step 4. If any of the informal requirements enters in conflict with any of the capabilities we granted to the attacker, and the channel used to transmit the element that has that requirement does not provide that security property either, then we have a security failure and we need to redesign our protocol. Specially, if for a given message exchange, any of the involved tokens has an informal requirement that cannot be met given the corresponding channel abstraction and attacker capabilities, there is a security flaw and it is necessary to go back to the initial steps and fix it. Otherwise, we have informally verified the design candidate and we can continue. However, this does not guarantee that the design will also pass the next phase of the methodology. In case we find a failure, we will feed back a list of informal requirements that have failed, in order to reconsider them in the next iteration of the methodology.

After applying our methodology up to this point, and assuming we passed the step 5, we would have reached the assurance level PAL1 defined in [142].

Phase II: Procedural verification of security. This part of the methodology is devoted to the procedural verification of the security requirements established before. It is depicted in the second phase of Figure 3.2. With procedural verification we refer to the fact that widely approved theories, methods or procedures should be applied here. Again, we can apply either the formal or the computational model. Nevertheless, we have found that the application of automatic (formal) tools (like ProVerif [39], Cryptyc [17], etc.) allows to detect many flaws with little effort, so it may be a good choice to first apply formal methods, and then use the computational model for a more concrete evaluation. Therefore, the steps we have included in this phase are as follows:

Step 6. Protocol pseudocode. In the same way that writing pseudocode is useful before coding a program, writing an informal narration of a protocol in the shape of pseudocode helps to reach a higher concretion level before properly formalizing it. As output of this step we get a written representation of the sequence diagram, with one process for each principal, which depicts the internal computations performed by each of them to generate the messages components along with the messages sent to the other principals.

Note that coding errors detected in step 8 might cause to come back to this step and modify the protocol's pseudocode (which will in turn induce changes in the code produced in the next steps).

Step 7. Formalization of the protocol. From the protocol pseudocode it is typically easy to produce a formalization in the language or definition model required by the chosen tools for verifying the protocol (the ones we decided to use in step 4). This formalization must include the entities taking part in the protocol, the communication channels and the attacker capabilities. Specially, two important aspects are worth to be emphasized in order to produce a reliable formal model. First, the channels that are employed in the model must be consistent with the

abstraction given in step 3. Second, the attacker and participating entities must also follow the assumptions made in steps 2 and 3, respectively. More concretely, if some entity is expected to behave dishonestly, it is necessary to model the attacker as if it would be able to act as/control that entity during the protocol. This may be simulated by a side-channel that leaks to the attacker all the private information known to the dishonest entity.

Step 8. Procedural verification. Using the formalization obtained in the previous step, we use it as input for the chosen procedural verification model or tool. Additionally, depending on the tool that we have chosen, we will need to determine which verifications we want to perform. For this purpose, we use the *informal requirements* produced at step 4. Specifically, in Algorithm 3.3 for each e subindex within the variable *req* we look for the last sub-step in which it appears (the one with greater s subindex). The value in $req[s][e]$ contains the formal security properties that we need to verify for the protocol element e (note that there may be elements with no requirements). Additionally, if the verification tool supports it, we can convert the *informal requirements* that are required to each element when it first appears in the *req* variable as a condition that needs to be ensured when that element is created in our formalization of the protocol. Note that even though the specific tool we use may not support this type of check, the system designers can always forward this requirements to the development team. That is, if the development team receives a requirement for an element t stating that it needs to be random and authenticated, they will easily deduce that they need to fetch it from a randomness source and digitally sign it. The process for obtaining the formal security requirements to be verified in this last step is shown in Algorithm 3.4. The properties required to each element when it is created can be obtained using a similar algorithm, but going through the *informal requirements* in increasing order.

```
Data: The informal requirements of Step 4.
Result: Formal security requirements.
forall the e in reqs do
    sreqs[e] = NULL;
    s = last step;
    while s >= first step AND sreqs[e] == NULL do
        if reqs[s][e] not empty then
            sreqs[e] = reqs[s][e];
        else
            s = s - 1;
        end
    end
end
```

Figure 3.4: Algorithm for assigning the final security requirements to be verified during the formal verification of the protocol. An entry to NULL in *sreqs* means that the corresponding element does not have any security requirement.

Finally, if the tools or procedures followed in this step "output" that all require-

ments are fulfilled, we can conclude. Otherwise, we have to go back to the first step and correct errors in our design, checking also for errors in the formalization. Like with the informal verification, if we find a security failure, we will feed back a list of the security properties that have failed, in order to reconsider them in the next iteration of the methodology.

Note that if a failure is detected during the first phase of the methodology, all the related information is fed back to be adequately treated in the first stage of the next iteration. However, when the failure is due to a coding error in the protocol formalization, the protocol designer would need to go back to the 6th step and revise the code without affecting the previous steps of the methodology.

As final comments on the methodology, we must point out that, by itself, the methodology does not give as output an explicit measure of security of the analyzed protocol other than checking whether the specified security requirements are held. That will depend on the tools or the models used to verify the protocol. In any case, we will obtain an answer to whether or not the protocol meets the requirements specified in step 4 of our methodology. According to [142], any protocol successfully verified using our methodology would reach an assurance level equivalent to PAL2 or PAL3, depending on whether the used tool provides bounded or unbounded verification[17]. Moreover, even the PAL4 level, still under consideration in [142], can be achieved if we use a computational model verification tool or method. Also, it is important to emphasize that, for protocols where several different execution sequences are possible, the methodology should be applied separately to each one of them. This includes error sequences, where the security properties of each element should be kept up to the point where the protocol has been executed. Note however that, if two different sequences share the same *sub-sequence* up to a certain point, and we have already verified that *sub-sequence*, it can be safely skipped. This may indeed be time-saving for the informal verification performed manually during the first phase of the methodology. Possibly, a good option (if the alternative sequences do not include additional factors, like new communication channels or special tokens) could be to only verify manually and formally the longest sequence. Once available its formalization, it would probably be easy to derive from it the alternative flows and formally verify them.

To summarize, the inputs and outputs of each of the different steps of our methodology are shown in Table 3.3 and Table 3.4, corresponding to the first and second phases of the methodology. The acronyms used in the tables for the different outputs are explained in Table 3.2.

3.3 Example analysis of real protocols

In this section we apply the different parts of the methodology to real life protocols. To show examples where flaws are found in both phases, we analyze two different protocols. The chosen protocols are EBIA's registration protocol, which presents a flaw detectable in the first stage, and the Shared Key Authentication of the WEP protocol, with a flaw that is detected in the second stage. For the latter, and for readability, we assume that the first phase has succeeded.

For a detailed application of the complete methodology to a protocol that succeeds in both phases, we refer to [85]. This system is analyzed using the present methodology

[17] As we saw in Section 2.3.1, unbounded verification means that the method is able to simulate attackers launching several instances of the protocol concurrently.

O1:	Informal list of goals.
O2:	Security model and +/- *capabilities list*.
O3:	Sequence diagram, channels abstraction and entities assumptions.
O4:	List of informal requirements / Requirement-tagged sequence diagram.
O5:	List of failed informal requirements or *Success*.
O6:	Pseudocode.
O7:	Formalization.
O8:	List of failed security properties or *Success*.

Table 3.2: Definition of the different outputs of the methodology.

	Goals	Attacker model and capabilities	Design candidate	Trust requirements	Informal verification
Input	None	[O5 or O8][a]	[O5 or O8][a] and O1	[O5 or O8][a] and O3	O2 and O4
Output	O1	O2	O3	O4	O5

[a]Only when we are not in the first iteration.

Table 3.3: Inputs and outputs of each step of the first phase of the methodology.

	Pseudocode	Formalization	Procedural verification
Input:	O8[a] and O4	O2, O3 and O5	O4 and O6
Output:	O6	O7	O8

[a]Only when we are not in the first iteration.

Table 3.4: Inputs and outputs of each step of the second phase of the methodology.

in [83], and we discuss the main results of this analysis in Chapter 5.

3.3.1 Informal analysis: Email-Based Identification and Authentication

In this section, we apply the methodology to the protocol known as *Email Based Identification and Authentication* (*EBIA*), which is almost certainly the technique that stands out among the set of alternatives for online registration[102]. As we will see, the methodology warns about a design flaw at the first stage (informal verification).

EBIA's popularity is due to its high usability and ease of deployment [103]. Briefly, EBIA works as shown in the sequence diagram in Figure 3.5 (recall the notation used for these diagrams, in Section 2.2). When a user wants to sign up in a web site in which she does not have a user, she sends the information required by the server to create a new one. In EBIA, this information contains at least the user's email address. The server responds to this request by sending an email containing an activation link. Upon receiving this email, the user accesses the link, fact that is interpreted by the server as a proof of ownership of the email address. Finally, in order to complete the registration process, the server creates an account for that user with the received information (typically, a username & password pair). In the best case, the first connection is a server authenticated SSL session; the second message is an unauthenticated, unen-

crypted email; and the third and fourth messages are both sent through a new server authenticated SSL session.

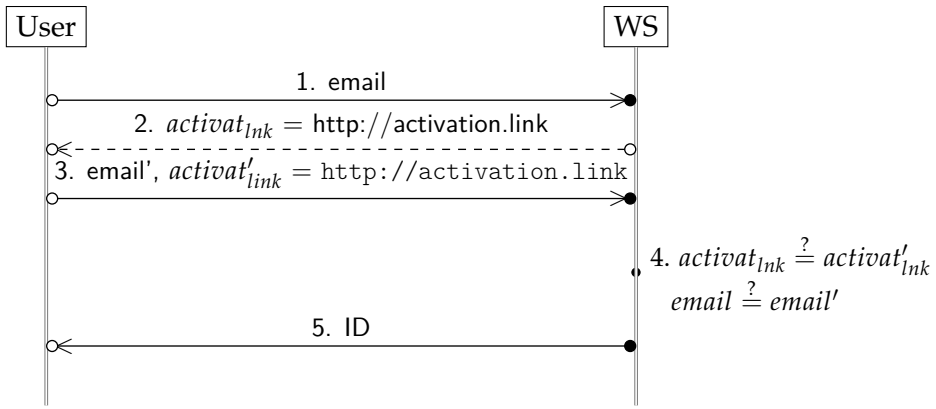


Figure 3.5: Sequence diagram of the typical EBIA protocol.

In summary, EBIA uses email addresses as identifiers and, as initial authenticators, the fact of accessing to URLs contained within email messages sent to those addresses. As we will see next, this process has major security problems, basically due to the fact that email addresses are an insecure means to send sensitive information. In this section, we apply the proposed methodology to EBIA, showing that this flaw in the assumption concerning email messages.

Step 1. Goals of the protocol. The aim of EBIA is to authenticate the registering user by means of her email address. Thus, the email address must be set as an identifying element unique and solely owned by the new user. Therefore, we set two goals:

1. Verify the ownership of the received email address.
2. Create a new identity associated to the received email address.

Step 2. Attacker capabilities and security model. In this initial and informal analysis, we start from a typical Dolev-Yao attacker. As such, she can arbitrarily intercept, block, replay messages, etc; but she cannot break cryptosystems, although may make use of them for creating keys, encrypting/decrypting arbitrary messages, etc. Besides this standard capabilities, we grant the attacker the ability to learn all the necessary information required in order to start a registration process in stead of any possible user. That is, in an hypothetical (but frequent) scenario in which, in order to sign up in a new website, users are required to provide their email address, birth date and real name, the attacker can obtain all those data. Note that this is a quite reasonable assumption nowadays, mostly due to the huge amount of information available from social networks and equivalent platforms[18]. As a result, our +/- capabilities list is in this case composed by just one element, namely: + *The attacker can obtain all the necessary information to start a registration process on behalf of any user*.

[18]The *OpenData* initiative, dumps of private data after security compromises, or even the documents published in the Spanish BOE (`www.boe.es`) are other interesting sources.

1. User $\leftarrow$ WS : $enc_{k_{SSL}}(\text{email})$
2. WS $\leftarrow$ User : email, $activat_{lnk}$
3. User $\leftarrow$ WS : $enc_{k_{SSL'}}((\text{email}', activat'_{lnk}))$
4. WS : email $\stackrel{?}{=}$ email$'$ and $activat_{lnk} \stackrel{?}{=} activat'_{lnk}$
5. WS $\leftarrow$ User : $enc_{k_{SSL'}}(\text{ID})$

Figure 3.6: Definition of the EBIA protocol using security protocol notation. WS stands for Web Server, and User represents the registering user.

Step 3. Design candidate, channels abstraction and entities assumptions. In a typical design process, we would have to come up with a design candidate at this step. However, in this example we already have a protocol to verify (the one depicted in Figure 3.5), which we depict with more detail using security protocol notation in Figure 3.6. We specially note that the email message (third message exchanged between the parties) is not encrypted.

As for the communication channels, we can distinguish the following types:

$\mathtt{A}_S\mathtt{C}$**.** This is the channel used for exchanges initiated by the new user. In particular, we assume that this are server authenticated SSL sessions, which is the most common case. Therefore, this channel offers confidentiality and server authentication.

P**.** This channel is used only once per protocol run, and it is implemented as an email channel. Specifically, the web site employs it to send the new user the activation link ($activat_{lnk}$ in Figure 3.6). It is well known that (conventional) emails do not offer point-to-point encryption, and no authentication at all. Thus, we consider this channels as potentially insecure.

Concerning the entities taking part in the protocol we assume that the web server is a trusted entity. This seems reasonable in the general case, since we consider that a web server would not obtain any benefit in allowing the creation of fake identities, since it would probably cause a reduction in the quality of its service, and an evident loss of reputation[19]. For users, we assume that they will not willingly allow the creation of illegitimate identities linked to their email addresses, since this would also damage their reputation (or worse). For that matter, we may see users as *honest-but-unwary* entities, who may leak personal information (like their email address, but not their credentials) by thinking that revealing this information does not pose any risk to them.

Step 4. informal requirements. Applying Algorithm 3.3 to the elements shown in Figure 3.6, we obtain the informal requirements specified in Table 3.5.

Step 5. Informal verification. We process now the previous informal requirements, taking into account the +/- *capabilities list* and security model produced at step 2, and the channels abstraction and design of step 3.

Message 1: The email address sent by the User to WS has "none" as informal requirement. Thus, this does not pose any requirement. Note that this as-

[19] Besides, very probably, being subject to legal risks.

Step in Figure 3.6	email	$activat_{lnk}$	email'	$activat'_{lnk}$	ID
1. User $\rightarrow$ WS : senc(email, k_{SSL})	none				
2. WS $\rightarrow$ User : email, $activat_{lnk}$	none	A_{WS}			
3. User $\rightarrow$ WS : senc((email', $activat'_{lnk}$), $k_{SSL'}$)	none	C, A_{WS}	none	none	
4. WS : email $\stackrel{?}{=}$ email' and $activat_{lnk} \stackrel{?}{=} activat'_{lnk}$	none	C, A_{WS}	none	none	
5. WS $\rightarrow$ User : senc(ID,$k_{SSL'}$)	A_{User}	C, A	A_{User}	C, A	C, A

Table 3.5: The rows under column *Figure 3.6* reference the corresponding step in Figure 3.6. A_E stands for tokens authenticated by entity E. A depicts tokens authenticated by all the entities involved in their transmission. C denotes a confidentiality requirement. *none* means that the token has already been sent, but without any assumption on its security properties. An empty cell informs that the token has not been sent yet.

sumption is due to the fact that the attacker may be the one initiating the registration (as stated in the capabilities list).

Message 2: The email address sent at the previous step is not modified. As for the activation link sent by the WS, it is assumed to be authenticated by the WS, since the SSL session established was server-authenticated, and confidential (it is a $\mathtt{A_SC}$channel). At this point, we already find two security flaws since the channel used to transmit the link is insecure. We could stop the analysis here, since this directly contradicts the expected informal requirements. However, for a better explanation of the issue, we continue until the last step.

Message 3: In this message, two additional tokens are sent $email'$ and $activat'_{lnk}$. At this point, they are not required any property, since may not be related at all with the previous ones.

Check 4: After this verification, if both the emails and activation links are the same, WS will consider that the user is the owner of the email address.

Message 5: As stated in check 4, when a match occurs, WS expects the email address to be authenticated by the user. Moreover, the activation link is interpreted as mutually authenticated by both the user and WS, since the latter was assumed to have been sent confidentially to the former. Consequently, WS trusts that only the legitimate user would have been able to access it. Again, due to the properties of the email channel (a Pchannel), these requirements do not hold.

In summary, the wrong assumption has the following implications.

1. The activation link does not satisfy the confidentiality requirement, given that it is sent through an insecure channel (a Pchannel, as defined in Section 2.2).
2. Therefore, an attacker with enough capabilities (e.g., in a privileged position) may, for instance: (1) initiate the registration on behalf of its victim, providing her email address, (2) intercept the activation link, and (3) complete the registration with the victim's identity by accessing the activation link. This attack is shown in Figure 3.7.

Consequently, the goal of "verification of ownership of the specified email address" cannot be satisfied since the activation link needs to be kept, con-

fidential, but it is being sent through an insecure channel. As we have seen, this is quickly detected as a design flaw during the informal verification of our methodology.

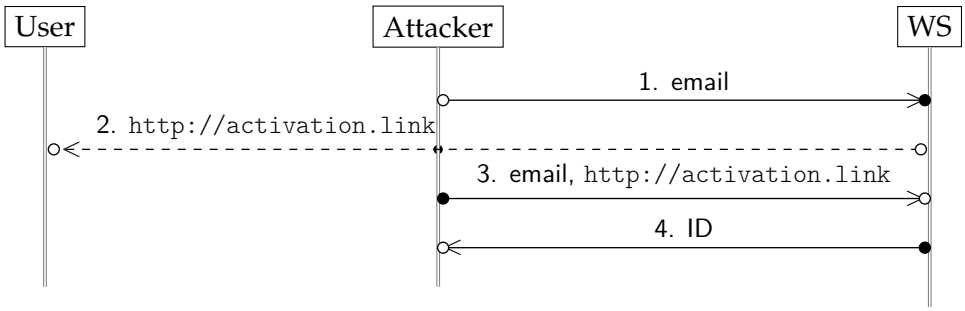


Figure 3.7: Sequence diagram of an active attack to the EBIA protocol. The attacker intercepts the email message containing the activation link. Even in the case in which the email is not blocked and is received by the user, she will most probably ignore it.

As consequence of this issue, the methodology forces to go back to the first step, and take this finding into consideration for the next design candidate. In Chapter 5 we propose a protocol following the same design principles than EBIA and keeping its usability, but that satisfies the expected security requirements.

3.3.2 Formal analysis: WEP Shared Key Authentication

We dedicate this subsection to the procedural analysis of the Shared Key Authentication of the WEP standard (WEP-SKA from now on), as defined in [119]. Applying the proposed methodology, we show how a well-known flaw is detected in the second phase (formal verification). For that matter, let us assume that the protocol successfully passes the first phase of our methodology. Also, in this case, we apply the automatic formal verifier ProVerif [39] for the formal verification.

The WEP-SKA protocol is one of the two authentication methods supported by the WEP standard. A normal execution consists of four messages between two stations: the Wireless Device (WD), and the Access Point (AP). In a typical protocol run the WD is authenticated by the AP as it is depicted in Figure 3.8.

Nevertheless, as pointed out in the WEP standard [119] sending both the challenge and its encrypted version may produce a security problem. This is due to the fact that WEP uses an encryption algorithm (the RC4 cryptosystem) that is a stream cipher which generates a pseudorandom sequence and XORs it to the plaintext in order to create the ciphertext. Therefore, if we know the ciphertext and the plaintext, we can obtain the keystream straightaway. However, in the standard it was only advised (but not required) to change the key and/or IV (Initialization Vector) frequently. As observed in [46], the fact of not being required to change the key/IV indirectly forces every receiver to accept repeated key/IV's, or risk otherwise not being compatible with some WEP compliant devices. That allows an attacker to successfully impersonate any station after having observed one single authentication. In [46] they call this attack *Authentication spoofing*.

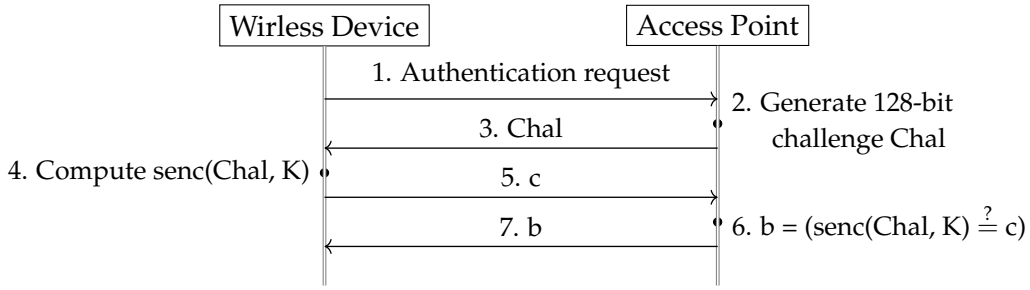


Figure 3.8: Message sequence of the WEP-SKA protocol [119]. (1) The WD requests to be authenticated using WEP-SKA. (2-3) The AP generates a challenge and sends it, in plaintext, to WD. (4-5) The WD encrypts the challenge with a preshared key K, and sends the result to the AP. (6-7) The AP checks the received encrypted challenge and informs WD of the result.

Let us now forget for a moment that this flaw is known, suppose that we are asked to verify the security of WEP-SKA, and that it successfully passes the first part of our methodology. Since we will need it in order to include it as a query for the formal verification, assume also that we have reached the conclusion that the *informal requirement* (produced as output of step 4) for the element c shown in Figure 3.8 is *authenticity* (which is inherited from the fact that the key K used to obtain c is a shared secret). Moreover, there is only one communication channel that is used during the whole protocol, and it is completely insecure (i.e., it is a `Pchannel` according to Section 2.2). The steps for the formal verification of the protocol are are as follows:

Step 6. Protocol pseudocode. The pseudocode for the WEP-SKA protocol is given in Figure 3.9.

Step 7. Formalization of the protocol. ProVerif has a problem here, because it does not support the XOR operation[20]. Nevertheless, since we do not need to apply complex derivation rules, we can get around this problem with a simple reduction rule simulating the XOR. This workaround can be seen in the code of the program available online [21]. It is a rule stating that if the attacker knows both the challenge c and its encryption (through a XOR function) with the key k, then she can apply the XOR function to recover the key k.

Step 8. Procedural verification. We have now formalized the protocol. Moreover, given that one of the *informal requirements* is for c to be authentic, we add an authenticity query within our ProVerif model. It is represented as a correspondence assertion stating that, each time the AP confirms having received a valid challenge response, a WD must have previously sent it. When we run ProVerif (the code for this example is available online[21],we observe a trace like the one shown in Figure 3.10, which informs that the attacker may authenticate to the AP as if it were a legitimate WD.

Consequently, the goal of establishing an authenticated session only between

[20] Although there are approaches for XOR-aware modifications of ProVerif. See [135].
[21] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

```
Process WD:
  send(MA,AUTH,(id,'shared key',1));
  receive(MA,AUTH,('shared key',2,info,result));
  if result == successful then
    chal = info;
    send(MA,AUTH,('shared key',3,senc(chal, k)));
  fi
  receive(MA,AUTH,('shared key',4,result));

Process AP:
  receive(MA,AUTH,(id,'shared key',1));
  if successful then
    result = successful;
    chal = new pseudorandom;
    send(MA,AUTH,('shared key',2,chal,result));
    receive(MA,AUTH,('shared key',3,senc(chal2, k)));
    if senc(chal2, k) == senc(chal, k) then
      send(MA,AUTH,('shared key',4,successful))
    fi
  fi
```

Figure 3.9: Pseudocode for the WEP-SKA procedures. *WD* stands from the Wireless Device to be authenticated, *AP* stands from Access Point and k is the shared key. The field *info* conveys information dependent on the authentication algorithm, and *result* stores the result of the requested authentication. *MA, AUTH, 'shared key'* and the numbers used in the messages are fields specified in the standard, see [119].

WEP-SKA between legitimate WD_1 and AP:
$WD_1 \rightarrow AP$: WD_1
$AP \rightarrow WD_1$: $Chall_1$
$WD_1 \rightarrow AP$: $enc_k(Chall_1)$
Attacker : $k = Chall_1 \oplus enc_k(Chall_1) \implies$ Attacker gains k
$AP \rightarrow WD_1$: OK
WEP-SKA between illegitimate $fake_{WD_2}$ and AP:
$fake_{WD_2} \rightarrow AP$: $fake_{WD_2}$
$AP \rightarrow fake_{WD_2}$: $Chall_2$
$fake_{WD_2} \rightarrow AP$: $enc_k(Chall_2)$
$AP \rightarrow fake_{WD_2}$: OK

Figure 3.10: An example attack trace found by ProVerif for the *Authentication spoofing attack* over WEP-SKA introduced in [46]. In the first block, the attacker eavesdrops on a WEP-SKA run between a legitimate WD and the AP. As a result, the attacker obtains the preshared key k. In the second block, the attacker uses the key k in order to successfully complete a WEP-SKA execution with the AP.

legitimate Wireless Devices and the Access Point[22] **cannot be satisfied**, given the fact that the attacker may obtain the preshared key K guaranteeing the authenticity of the exchange. Therefore, this is a design flaw that has been detected during the formal analysis of the methodology.

Since we have found an attack during the procedural verification with ProVerif, the requirements are not fulfilled. Therefore, we must go back to the first stage of the methodology after checking that no *coding* errors have been made, and re-design the protocol to avoid this attack. Moreover, since the specific security requirement that has failed is the authenticity of the challenge c, this is the component that we need to revise (along with everything interacting with it). For instance, we could proceed like it is suggested in [46], namely, we could disallow the reuse of IVs.

3.4 Chapter conclusion

In this chapter, we have pointed out the necessity of procedures to formally define a security context from a set of informal requirements and properties enabling a formal verification of communication protocols. We also pointed out the need to reduce the gap between a theoretical level design and the practical level equivalent.

For the first task, we have defined an iterative methodology that starts with an informal definition of each component that needs to be taken into account, and then converts those informal definitions into formal ones. Additionally, by performing first an informal verification, and subsequently a formal one, the methodology detects simple flaws at the early stages, thus saving costs of having to formally redefine everything for issues that can be easily detected.

For the second task, aimed to reduce the gap between theoretical designs with practical ones, our methodology incorporates fine-grained capabilities for the attacker. As a result, the produced security model may be adapted as needed to any possible context that may arise in the real world, hence producing veracious models.

We emphasize once again that, even though we have shown here how to apply the methodology using automated verification tools (and more specifically, ProVerif), it is perfectly compatible with any other tool and with proofs of security based on the computational model. Moreover, the application of the methodology in conjunction with other tools and techniques will certainly offer deeper insight into what additional features need to be taken into account. Indeed, the methodology presented in this chapter has been refined over time by applying it to several protocols and systems (among them, the ones in the following chapters of this thesis). Therefore, further applications of it will undoubtedly help in further improving it. Specially, applying different tools and theories in the second stage would provide useful insight as to what aspects are necessary to take into account, informally, during the first stage in order to produce better design candidates and ease the task of subsequently formalizing them.

In addition, the work of [96] may be pointed out as a notable future line of work, in the sense of further expanding our methodology towards easing the task of formalizing security requirements that are otherwise hard to state in a formal manner. This proposal differs from the rest in that it takes into account the *human factor* at the time of formally verifying the security properties of a system. Indeed, this is usually ig-

[22] Even the goal has not been explicitly stated in this example, since we just analyzed the second phase of the methodology, it is straightforward that this would be the main goal of the protocol.

nored, while on the other hand it is the main source of security problems. With the concept of *ceremonies*, and by representing users as separate entities in the protocol, the work in [96] allows to include them into a formal model, suitable for the previously mentioned verifications. For instance, the concept of *ceremonies* allows modeling social engineering attacks.

Finally, besides the examples given in this chapter, we show how the methodology is employed in subsequent chapters. Namely, in Chapter 5 we apply it to SEBIA, our secure alternative proposal to the already studied EBIA protocol. Also, the system described in Chapter 7 is verified in Appendix B with this methodology, including formal and computational approaches in the second stage.

CHAPTER 4

libgroupsig: An extensible C library for group signatures

Chapter based on and supported by references [21, 23, 86].

While the general principles of cryptographic primitives may be clear to programmers, their internal details are not usually so well understood [101]. In addition, most theoretical proposals in cryptography are oriented to very specific scenarios which makes difficult their inclusion in a wider set of practical contexts. Certainly, in most cases where a software implementation is provided to backup a theoretical contribution, it is very difficult to adapt it and (re-)use it in more complex systems demanding the functionality that this software provides. An example of this situation is depicted by the creation and use of group signatures schemes [65].

In order to overcome this problem, we have adopted the recommendations in [153], and thus we have analyzed the most relevant works in the field of group signatures to extract the underlying principles and the involved functionality. Correspondingly, we have designed an Application Programming Interface (API) and implemented a prototype using C programming language. Regarding the implementation, we have used three group signature schemes as bottom line. Nevertheless, in the API design we have taken into account that specific group signature schemes may bear only a subset of the functionality associated to the group signature primitive. As a result, our library supports the addition of new schemes without the need of modifying the existing code.

This library has been actually used for implementing the prototypes of the different proposals that have been described in this work. Specifically, those in Chapter 6 and Chapter 7. But, more importantly, on the basis of the main conclusions in [116], we have made our implementation open source[1] in order to enable its use in advanced privacy respectful systems, and to promote its revision and improvement through the collaboration of the whole PETs community.

In Section 2.4.1 we made an overview of previous work in group signatures primitives, including a summary of the main operations present in these schemes. These main operations are what we have used as a base for constructing the API of our library, which is described in the following sections.

[1] Available at `https://bitbucket.org/jdiazvico/libgroupsig/`.

4.1 Related work

Group signatures have been standardized by the ISO/IEC: [120], defines the general setting and main operations of group signatures[2]; [121], defines a total of 7 schemes with opening and linking capabilities. In [31, 82], extensions to the X.509 Public Key Infrastructure [185] are proposed in order to be able to deal with identities based on group signatures (as we will see in detail in Chapter 6).

Several implementations of group signature schemes are currently available online. BBS04 [44] is implemented in C within the `PBC_sig` library[3] and using Python within the Charm framework[4] [13]; BCLY08 [32] is implemented in C in the `FTMGS` library[5]; CG04 and ACJT00 are implemented in the `libgs` library using Java, as part of the PP2db project[6]. Finally, the framework in [164] is intended to analyze and compare the performance of different schemes. The framework is written in Java, and also implements CSST06 [59], BCC04 [50] and IMSTY06 [124].

As we describe in the following sections, our `libgroupsig` library[7] implements BBS04 [44], KTY04 [132] and CPY06 [67]. But more importantly, it allows the addition of new group signature schemes through the same API. This would certainly ease the use of group signatures as a building block of larger systems, since the costs of changing from one scheme to another would be reduced to a minimum.

4.2 Group signatures API

Next, we briefly describe the general interface that we have defined for interacting with the functionality provided by the `libgroupsig` library, according to the main operations introduced in Section 2.4.1. Figure 4.1 shows an UML-like class diagram, that depicts the described API in a component-wise manner. The main component is the one aimed to the interaction with group signature schemes, named `groupsig` in Figure 4.1. The functions defined within this component are:

`groupsig_init`. Initializes the library environment, including the internal Pseudo Random Number Generator.

`groupsig_clear`. Frees the internal variables initialized in the previous function.

`groupsig_setup`. Initializes the scheme with the specified code, filling the group key, manager key and GML. Uses the input parameters contained in the specified configuration structure for controlling the generation process.

`groupsig_join_mem`. Executes the join member part of the scheme. The member key will be updated with the member side generated keying information. Note that, in most schemes, there is typically a member side and a manager side of the join process, which may be used to prevent the manager from learning private tokens. If, nevertheless, the manager runs all the joining functionality, this function could just be left as a stub.

[2] The document differentiates between group and ring signatures, using the general term anonymous signatures to refer to both of them.

[3] `http://crypto.stanford.edu/pbc/sig/`. Last access: January 2nd, 2015.

[4] `https://code.google.com/p/charm-crypto/`. Last access: January 2nd, 2015.

[5] `http://www.lcc.uma.es/~vicente/swprj/index.html#libftmgs`. Last access: January 2nd, 2015.

[6] `http://www.ing.unibs.it/ntw/tools/pp2db/`. Last access: January 2nd, 2015.

[7] https://bitbucket.org/jdiazvico/libgroupsig

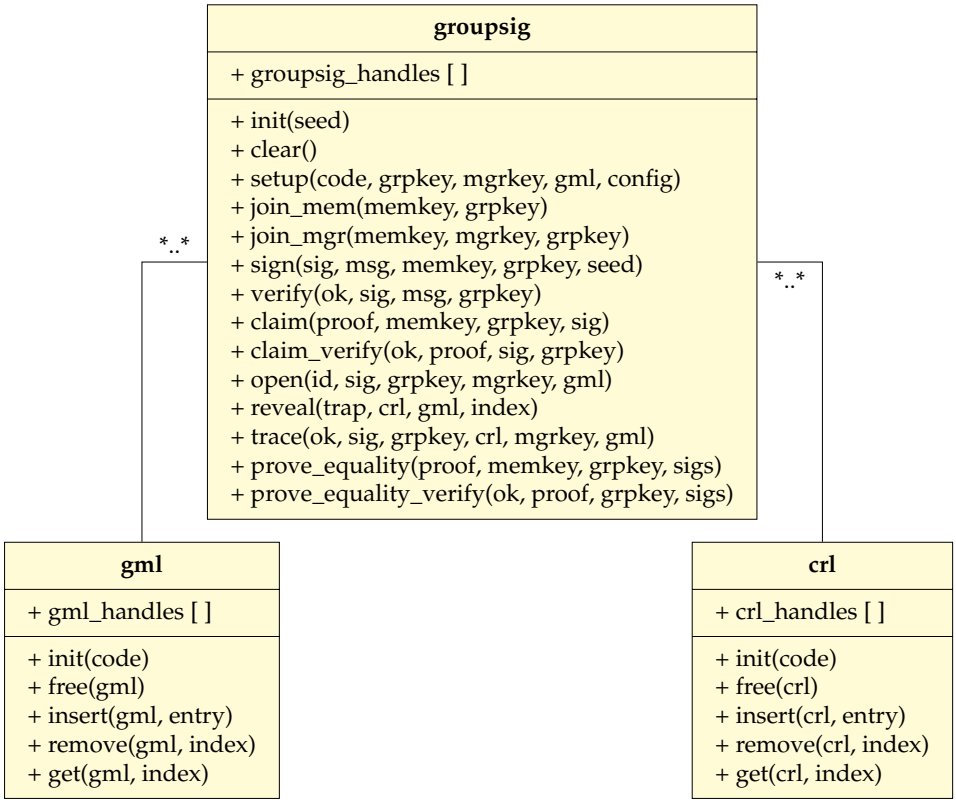


Figure 4.1: UML-like class diagram for the `libgroupsig` API. For readability, we omit variable types.

`groupsig_join_mgr`. Runs the manager side of join of the scheme. With it, the member key is completed, and a new entry related to the new member is added to the GML.

`groupsig_sign`. Runs the signing algorithm of the scheme and stores the resulting group signature in `sig`. The `seed` parameter is useful when reseeding the internal Pseudo Random Number Generator is necessary.

`groupsig_verify`. Verifies the given signature with the received message and group key.

`groupsig_claim`. Issues a zero-knowledge proof claiming having issued the specified signature for the given group and member keys.

`groupsig_claim_verify`. Verifies whether the given claim is correct for the specified group signature and group key.

`groupsig_open`. Returns the real identity of the issuer of the given signature. In our library, this may imply the addition of the identity into a CRL for members with revoked anonymity.

`groupsig_reveal`. Reveals the tracing trapdoor of the member in position `index` within the given GML. In our library, this may imply the inclusion of the tracing trapdoor into a CRL for members with revoked unlinkability.

groupsig_trace. Determines whether or not the issuer of the specified signature has been revoked according to the given CRL.

groupsig_prove_equality. Creates a proof of equality of all the group signatures within the given set, using the specified member and group keys.

groupsig_prove_equality_verify. Verifies the given proof of equality, associated to the specified set of group signatures.

Note that our API is basically equivalent to that of [120] (our *reveal* function is equivalent to their *revoke* process, and our *trace* function is equivalent to their *link* process). However, we also include the *claim* and *equality prove* actions, which allow operations not directly supported by the ISO/IEC standard, but that are certainly important in many contexts.

Besides the core functionality for group signatures, the library also includes two components for managing Group Membership Lists (GMLs) and Certificate Revocation Lists (CRLs). Also, for the sake of achieving a uniform API for all the schemes, we have not been completely strict on some matters. The following subsections summarize this.

4.2.1 GMLs and CRLs

The modules `gml` and `crl` are intended for the management of Group Membership Lists (GMLs) and Certificate Revocation Lists (CRLs), respectively. GMLs are lists of members, typically set up during the group initialization and updated each time a new member is added (or permanently removed). They contain important information that may be used when either anonymity or unlinkability revocation is required. CRLs are named after their equivalents in the X.509 infrastructure [185], but in this case they are employed within the extended setting created by group signatures [82]. That is, they are intended for keeping a list of member keys for which either their anonymity or their unlinkability properties (or both) have been revoked (see Section 2.4.1).

The main operations provided within the `gml` and `crl` components are the ones typical of a list-like structure. Therefore, we allow the creation and liberation of these structures, the insertion (resp. removal) of new (resp. existing) elements through the insert (resp. remove) action, and the access to elements in the list through the get action.

4.2.2 Implementation notes

In the library, we have followed the abstraction outlined in Section 2.4.1. However, not all group signatures actually provide the same functionality set. For the sake of a more unified API, we have been slightly loose when assigning names to each function, and simultaneously we have been cautious to avoid misleading potential users of our library.

For instance, in KTY04 and CPY06 there are two revocation functions: *open*, which given a group signature and (part of) the join transcript of a group member (stored within the Group Membership List in `libgroupsig`), returns the real identity of the signer; and *reveal*, which given (part of) the join transcript of a specific group member, returns his tracing trapdoor. In the library, we refer to the respective parts of the join transcripts as *open trapdoor* and *tracing trapdoor*. However, BBS04 does not natively provide the same revocation options like that of KTY04 or CPY06, since it does not contain what we call tracing trapdoors. Nevertheless, tracing is still possible in BBS04, although in a less privacy respectful way. Indeed, what it is called *tracing* in BBS04

implies executing the *open* procedure (thus obtaining what we named *open trapdoor*) and looking for the signer's identity within a list of revoked members. Thus, it is not actually precise to use the term *reveal* with BBS04 to refer to the procedure defined with this name in KTY04 or CPY06. Nevertheless, in `libgroupsig` we use the term *reveal* to name a procedure that, given the part of the join transcript of a specific group member used as tracing trapdoor, includes it within a CRL, which will be subsequently used for tracing. This allows us to create a unified API for similar functionality, although the inner cryptographic details (and privacy implications) may not be equivalent.

The library also contains additional modules for implementing functionality not directly related to group signatures, GMLs, or CRLs. This code is basically divided in mathematical functions (mostly some number theory algorithms) in a module named `math`; the `sys` module, which defines system-wide functions such as memory management functions, global constants and environment variables; and the `misc` module, which implements functionality for reading and printing information, type conversions, etc. There is also a component of the library for the management of group member identities in an abstract manner, i.e., for making GMLs independent on whether the programmer wants to include full names, job positions, etc. within the data stored in each GML entry.

`libgroupsig` is available under GNU LGPL at Bitbucket[8]. We have tested and applied it in our prototypes, but its development is still in an alpha stage and, by opening its source to the community, we expect to receive useful feedback to improve it.

4.3 Experimental evaluation

The acceptability of a software library is very dependent on the functionality achieved, but also on the efficiency of the final implementation. Therefore, after defining the interface, we have implemented it using the C programming language in order to analyze the costs associated to each of the supported actions explained in Section 4.2[9]. Since BBS04 and CPY06 use elliptic curve cryptography while KTY04 is RSA-based, we show the costs associated to key sizes providing roughly the same security level. Specifically, according to the NIST[10], the equivalences are as shown in Table 4.1. All the measurements have been obtained with a desktop PC (Intel Core i7-2600, 16GB DDR3 running Debian Wheezy), iterating 1000 times for each operation and using different keys in each iteration.

RSA key size	ECC key size
1024	160
2048	224
3072	256
7680	384

Table 4.1: Approximate key sizes providing equivalent security for ECC and RSA schemes.

[8] `https://bitbucket.org/jdiazvico/libgroupsig/`.

[9] We do not include measurements for the *prove equality* functionality (and its verification counterpart), which is a generalization of the claim action (resp., claim verify action). Thus, the specific cases give a good idea of the costs related to their more general counterparts.

[10] `http://www.nsa.gov/business/programs/elliptic_curve.shtml`. Last access on March 31st, 2015.

Figures 4.3 through 4.6 depict the costs associated to each of the main operations excluding tracing, for each of the implemented group signature schemes in our library. In all cases, the evolution starts to differ notably for keys larger than 3072 bits (for KTY04) and keys larger than 256 bits (for BBS04 and CPY06), with differences of at most a few tenth of seconds for smaller keys. Specifically, for keys of size 7680 bits the costs of KTY04 increase abruptly, while the equivalent in BBS04 and CPY06 (384 bit keys) maintain a reasonable growth. The increase in KTY04 is most probably due to the costs associated of operating with larger numbers. Indeed, the three schemes rely on GNU GMP[11] (KTY04 uses it directly, while BBS04 and CPY06 through Ben Lynn's PBC library[12]). To verify this, we performed a profiling of GMP, based on the size of the employed numbers. Figure 4.2 shows the result. The profiling of GMP was performed in the same system as the one used for the analysis of libgroupsig. The tests involved 1000 iterations of GMP numbers ranging from 1000 to 10000 bits, increasing 100 bits per iteration (X-axis). Each iteration includes the basic operations: addition, multiplication, exponentiation and random number selection. The values in the Y-axis are the average running time for each iteration. It can be seen that the evolution of the CPU time in the profiling of GMP and that of the group signature schemes confirm our hypothesis.

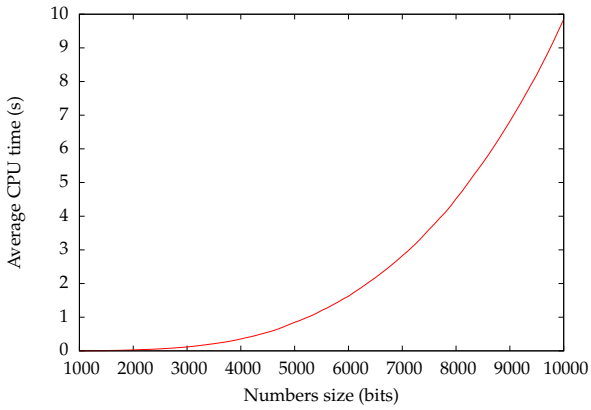


Figure 4.2: Profiling of the GMP library. The increase in the costs of computing with GMP, depending on the size of the numbers is reflected in KTY04 in the Join operation (as well as in the operations shown below

For analyzing the costs of tracing, we need to consider both the size of the Certificate Revocation Lists (CRLs) and the keys. The graphs in Figure 4.7 show the evolution of the associated costs, given these parameters. BBS04 is by far the most efficient one (nevertheless, consider the observation made in Section 4.2.2), with costs always less than 0.008 seconds; KTY04 is also quite efficient up to keys of 3072 bits, but increases steeply from less than 5 seconds per tracing operation to more than 20 seconds when using keys of 7680 bits; finally, CPY06 is the most expensive in this operation, growing uniformly depending on the key and CRL size from 2 seconds to 18 seconds per tracing operation.

Besides the tests summarized here, libgroupsig has also been employed in other research works where group signatures play a central role. Namely, we have used it for implementing a proof of concept of the extensions to X.509 in [78, 82] described in Chapter 6, and to build up a prototype of the privacy respectful online shopping

[11] https://gmplib.org/. Last access on March 31st, 2015.
[12] http://crypto.stanford.edu/pbc/. Last access on March 31st, 2015.

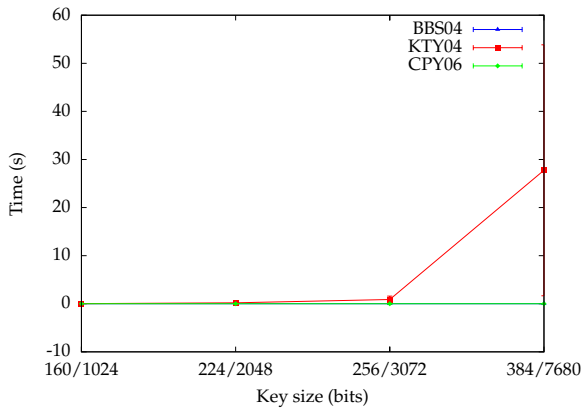
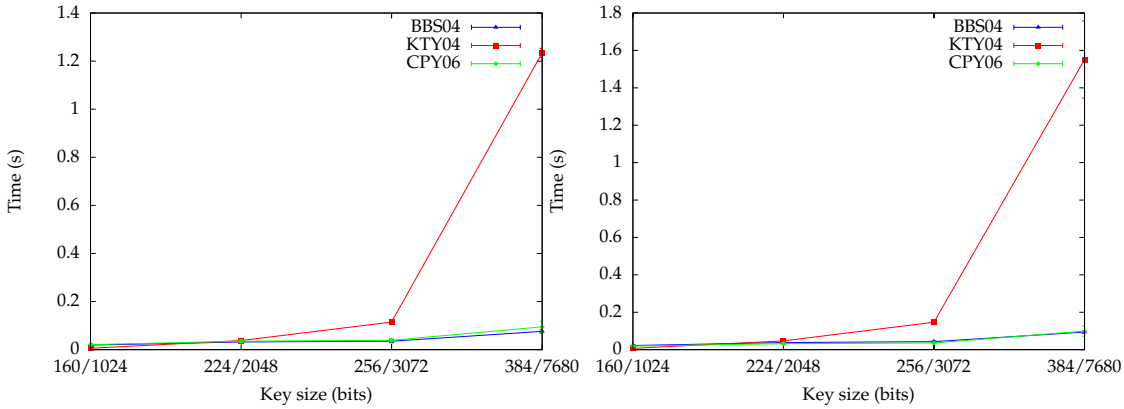


Figure 4.3: Costs of Join in KTY04, BBS04 and CPY06 (the lines for BBS04 and CPY06 mostly overlap).



(a) Sign. **(b)** Verify.

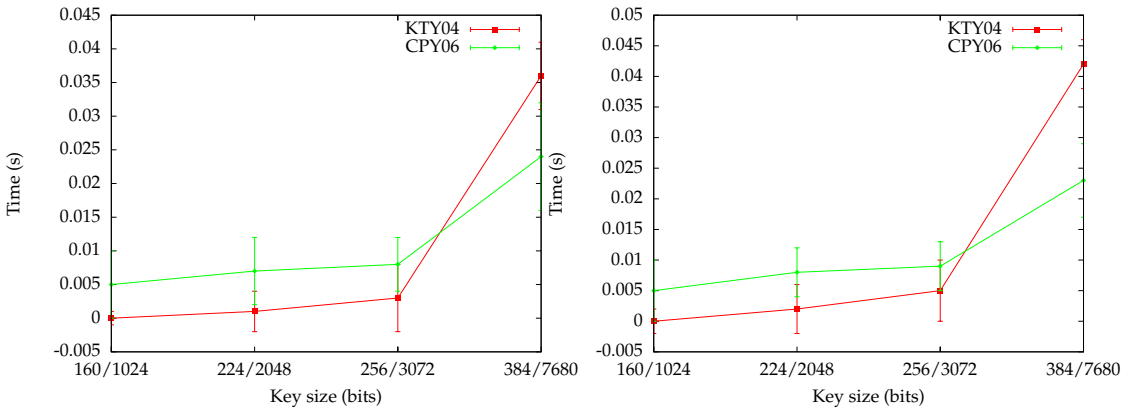
Figure 4.4: Costs of Sign and Verify operations in KTY04, BBS04 and CPY06 (the lines for BBS04 and CPY06 mostly overlap).

platform in Chapter 7

4.4 Chapter conclusion

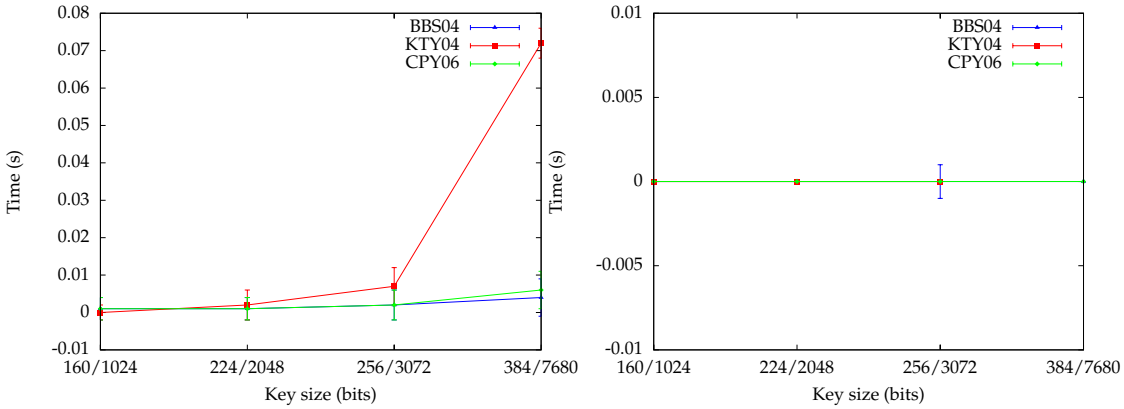
We have presented `libgroupsig`, a C library that provides a uniform API for different group signature schemes. Moreover, the library supports the addition of new group signature schemes without needing to modify the already implemented code. This offers very interesting possibilities. For instance, new schemes may be seamlessly incorporated into our library (see Appendix C); or complex systems where privacy is of concern may use our library with a group signature scheme A, but switch to another scheme B if needed with just updating a few tenths of lines of code at most (see Appendix C). To the best of our knowledge, no existing open source library provides equivalent possibilities.

Given the usefulness of group signatures as a building block for providing privacy, and the features of our library, we consider that our contribution may help in



(a) Claim. **(b)** Claim Verify.

Figure 4.5: Costs of Claim and Claim Verify operations in KTY04, BBS04 and CPY06 (the lines for BBS04 and CPY06 mostly overlap).

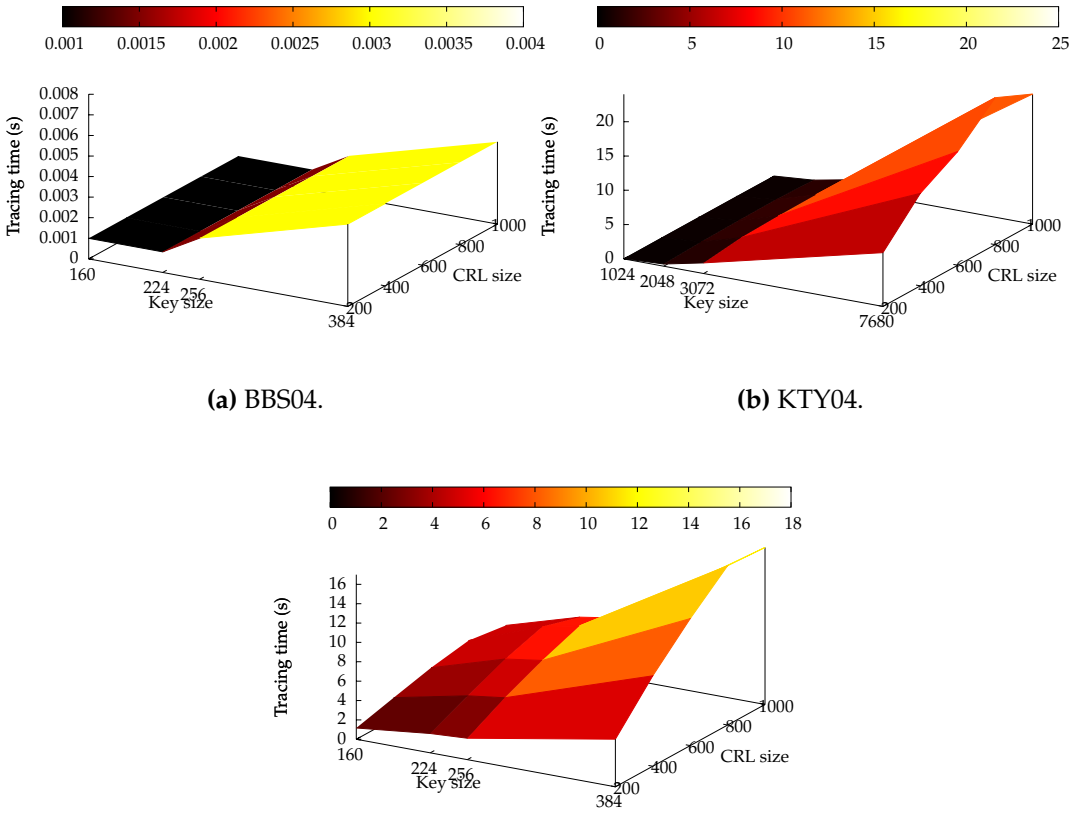


(a) Open. **(b)** Reveal.

Figure 4.6: Costs of Open and Reveal operations in KTY04, BBS04 and CPY06 (the lines for BBS04 and CPY06 mostly overlap).

the development of advanced privacy respectful systems. In the past, open source libraries corresponding to advanced cryptographic primitives have served as catalysts for prototypes and complex cryptographic systems (see, e.g., the PBC library and all the projects that depend on it[13]). Keeping in mind the future work still pending we expect that our library might thus help in the creation of new advanced privacy respectful systems. The possibilities are many. For instance, we have already employed our library to implement the prototype of a comprehensive and privacy respectful online shopping system, suitable for current e-commerce infrastructures. Also, one of the final applications we have in mind for this library is to conform the basis for X.509 extensions like the ones proposed in [31, 82]. This would undoubtedly suppose a great improvement of the X.509 PKI [185] towards supporting privacy respectful systems and applications. In [78, 82] we already applied this library for implementing a proto-

[13] `http://crypto.stanford.edu/pbc/who.html`. Last access on March 31st, 2015.



(a) BBS04.

(b) KTY04.

(c) CPY06.

Figure 4.7: Costs of Trace in `libgroupsig`, for increasing key and CRL sizes. The color gradient depicts time in seconds.

type of the mentioned extensions.

Nevertheless, as with every programming project, despite the library has reached a fully functional state, more work is required to improve it. First, the library is still in an alpha stage, and much testing is necessary for guaranteeing its correct functioning, including tests in as many different platforms as possible. In addition, the implementation of the currently supported schemes could probably be improved (through code optimization) towards achieving better efficiency, and the inclusion of more schemes in the library will help in testing and refining its extensibility, and also help creating a richer range of schemes to choose from. For instance, schemes where revocation is based on accumulators [55], in which tracing is more computationally efficient (at the cost of higher communication costs); or the more efficient schemes in [137]. Also, adapting our library to suit the testing framework in [164] would help in automating the comparison of the implemented schemes. Finally, given that the final aim of this library is to be employed within cryptographic systems, a source code security audit is mandatory. In this regard it is relevant to underline that the publication of the library as open source contributes not only to its inclusion in more complex projects, but also its evaluation through the open source community.

Finally, note that with this library, we aim to provide practical means to implement

actual privacy preserving systems. As stated, anonymous identities like those managed in Chapter 6 could actually be implemented through our library, but also any other token based on group signatures, like the ones employed in Chapter 7.

Part III

Protocols

Part III includes two protocols and two mechanisms that are essential for any system requiring robust authentication, which have also been created with the help of the methodology and group signatures library in Part II. It is worth to be noted that these contributions are based in the most widely used technologies for the tasks they address. Specifically, in Chapter 5 we describe a registration protocol based on the Email Based Identification and Authentication protocol (EBIA). Our proposal allows, assuming that new users own email addresses in trusted email providers, to create cryptography-based identities with security guarantees that are reasonable for many general contexts. This protocol has been designed through the specific application of the methodology described in Part II. Chapter 6 includes extensions to the OCSP (Online Certificate Status Protocol) and CRL (Certificate Revocation List) mechanisms for managing anonymous X.509 identities based on group signatures. Additionally, given that providing evidence of illegitimate actions is a critical task in systems providing fair anonymity, we also propose a protocol for this purpose. These mechanisms have been tested through a prototype implemented using the group signatures library described in Part II. Standard-like definitions of these mechanisms are given in Appendix A. Finally, these protocols are completely compatible with the systems proposed in Part IV.

CHAPTER 5

SEBIA: Secure Email Based Identification and Authentication

Chapter based on and supported by references [77, 83, 85].

The first step that every user must perform in order to start using any web-based system is to register in it. These registration protocols have as main objective the creation of some type of credential, which will subsequently be responsible for ensuring that no unauthorized user accesses restricted information. Therefore, the security properties that may be provided by the service are directly dependent on the security properties of these credentials. This is, by itself, enough reason for justifying additional efforts into the development of secure registration protocols. Additionally, the huge volume of users that Internet services handle nowadays, along with the increasing amount of information stored within them and its sensitivity, claims for further security guarantees at this point. To show some miscellaneous data, Dropbox reached the barrier of 275 million users[1], Facebook has approximately 680 million active users[2] and Twitter about 289 million[3]. Each of these users has registered using the EBIA protocol studied in Chapter 3. Moreover, we note that systems that use mobile phones as authenticators, instead of email addresses, follow exactly the same guidelines. In this variant, instead of sending an email to the specified email address, a Short Message Service (SMS) message is typically sent to the given mobile phone. However, the channel used to send SMS messages, if mobile phones are used as the unique authentication mechanism, provides roughly the same security level than an email since SMS messages are not protected by point-to-point encryption [97]. This observation is important, since given the increasing presence of smartphones[4] this variant is gaining impulse (although mobile-based authentication schemes and mobile-based two-factor authentication still need some refinement [92]).

Thus, secure registration protocols are required. Nevertheless, usability must be guaranteed too or, given the dynamic nature of the Internet, an unfriendly protocol will just be ignored. As stated, probably the most currently widespread registration protocol in online systems is EBIA, which is also known to be insecure (see Section 3.3.1). Thus, the identities generated by means of this protocol, are also fundamentally

[1] `http://techcrunch.com/2014/04/09/dropbox-hits-275m-users-and-launches-business-product-to-all/`. Accessed on March 31st, 2015.

[2] `http://www.statisticbrain.com/facebook-statistics/`. Accessed on March 31st, 2015.

[3] `http://www.statisticbrain.com/twitter-statistics/`. Accessed on March 31st, 2015.

[4] `www.technologyreview.com/news/427787/are-smart-phones-spreading-faster-than-any-technology-in-human-history/page/2/`. Accessed on March 31st, 2015.

insecure from the perspective of their authentication guarantees. In this chapter, we propose Secure Email Based Identification and Authentication (SEBIA), a secure alternative maintaining the same design principles than EBIA (specially, its usability) but also providing a security level that would be enough for many Internet-based applications. With this proposal, we place an additional piece in the puzzle for composing secure online systems. Concretely, this protocol is designed and verified using the methodology proposed in the previous chapter and is perfectly suitable for the generation of the robust cryptographic identities used for implementing fair anonymity in the next chapters.

5.1 Related work

Several alternatives have been proposed to solve or reduce the impact of EBIA's security limitations, typically by incorporating some kind of multifactor and/or multichannel method [184]. In [182] single use authenticators provided through a multichannel technique are employed with the aim of reducing the effect of possible attacks. The security of the scheme is claimed to improve the performance of EBIA by adding a token sent via SSL during the same session established when the registering user initiates the authentication request. Besides sending this token via SSL, an extra token is sent unencrypted via email. The related digital identity is only generated if the user sends back to the server both tokens. The intuition behind this approach is that, by using SSL, only the user who started the authentication request receives the SSL token. Thus, if this user also proves knowledge of the token sent via email, it is assumed that the user who started the session is the owner of the email account. However, even with SSL in use, this approach is vulnerable to the attack on EBIA illustrated in Figure 3.7 (attack initiated with step 1b). Although it is an attack that assumes that the attacker takes an active role (i.e., it is not limited to just observing the communications), it shows that the protocol's security is not higher than that of classic EBIA. In detail, an active attacker starting the registration process would just have to eavesdrop the activation email to obtain a legitimate activation code, and then use it. For the sake of clarity, the code used for showing this attack is available online[5] This shortcoming is addressed in [182] by combining several email accounts. In this new setup, the challenge for an attacker is more complicated, since she has to break into several mail servers. Nevertheless, this also erodes the scheme's usability as users have to manage more than one email address, and thus they can be reluctant to adopt this alternative. As a result, a third possible implementation is proposed in [182] consisting first of the encryption of the email using a key sent jointly with the SSL token, and secondly by routing the resulting message via an anonymizing network. Therefore, even though the attacker can start the SSL session and get the SSL token plus the key used to encrypt the email, allegedly she cannot gain possession of the email because it is sent via an anonymizing network. However, if the communications with the exit (entry) point from (to) the anonymizing network are not protected, this option is still insecure.

In [9] EBIA takes an important role in a method to avoid phishing attacks[6]. The method is a two-factor authentication procedure based on browser bookmarks and fragmented identifiers. However, the setup of these bookmarks relies directly on a verification code sent to the registering email address. To secure this step, the author states that this email should be "secure in authentication and in content" (hence, signed and encrypted). Otherwise, it would also be vulnerable to the same attacks as EBIA. Indeed, encryption and sender authentication in the activation email is a valid option (in fact, it would probably be the best one). Nevertheless, we prefer to avoid it, mainly because of two reasons: first, because users seem reluctant to use email encryption [104, 173, 183]; and second, if the emails can be sent in such a way then the authenticating token can be sent straight away within the email. Certainly, only the legitimate owner of the email account is going to be able to read it (this is guaranteed by encryption), and thus any extra step to provide her with a valid token seems unnecessary.

[5] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`.

[6] As defined in [167], phishing attacks are *"malicious emails that victims receive effectively convincing them to visit a fraudulent website at which they are tricked into divulging sensitive information"*.

5.2 The protocol

We proceed to explain the proposed protocol, introducing its main objective and the entities that take part in a normal protocol run.

5.2.1 Protocol description

This protocol is intended to establish a mutually authenticated channel between users and a Web Server (this would be a Achannel, according to Section 2.2), suitable for a subsequent distribution of digital identities. In particular, the users' identities will be associated to their email addresses. Indeed, this is what the protocol must actually ensure: that the user claiming to be the owner of a given email address is actually its owner. This will subsequently enable the distribution of a digital identity associated to that email address.

Entities. There are three different entities in our proposal:

- **Users**. The entities who start the registration process. Each one is assumed to own an email account with a trusted email provider.
- **Mail Server (MS)**. A Mail Server of the domain where the user owns an email account. The Mail Server is assumed to own a digital identity trusted by the Web Server.
- **Web Server (WS)**. The Web Server providing the registration service.

Protocol sequence. The process starts when a user requests to be registered in a web site. In our protocol (as usual) this is done by establishing a server-authenticated SSL/TLS session with the WS and having the user send her email address. Then, the WS uses the same SSL/TLS session to send back an *SSL registration ticket* (which must be unique to this request) and a nonce, named *email registration nonce* (henceforth, *email nonce*). Furthermore, the WS sends the same *email nonce* to the user through the MS as an unencrypted email message. Subsequently, the user securely connects with her MS and waits for the email. When it arrives, she verifies that the nonce received via email matches the one received via SSL/TLS. If so, the user sends the MS a request to acknowledge the receipt of the message. Consequently, the MS digitally signs the ACK, and sends it to the WS. At this point, the WS has the certainty (because it trusts the MS) that the legitimate owner of the email address has received and acknowledged the email with that concrete nonce. Finally, when the user sends back the SSL/TLS registration ticket, the WS issues the new user's credentials.

A general view of the protocol, using the protocol notation for sequence diagrams described in Section 2.2, is shown in Figure 5.1. In the next section we proceed with an in-depth security analysis of the proposed protocol.

Note that, in short, there are two main modifications over EBIA. First, the token sent via SSL and the fact that the email token is also sent within the same SSL session. The second modification is the explicit endorsement made by a trusted entity (in this case, the Mail Server) over the identity of the user. However, it does not do it blindly, since the user has previously authenticated against it. Intuitively, this is assumes a transitive property in the trust relation: since MS trusts the user, and WS trusts MS, then WS trusts the user.

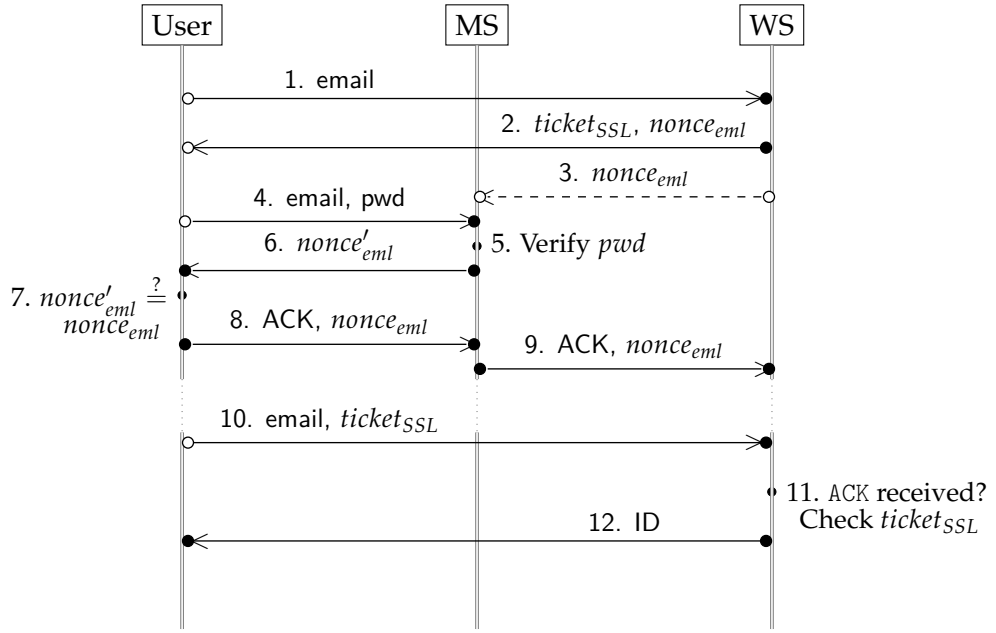


Figure 5.1: Sequence diagram of a normal run of the protocol. User stands for the user requesting registration, MS stands for Mail Server and WS stands for the Web Server providing the registration service.

5.3 Security analysis

Next, we discuss the verification of the security properties of the registration protocol according to the methodology introduced in Chapter 3.

5.3.1 Informal verification

The five steps comprising the first stage of the methodology are as follows.

Step 1. Goals of the protocol. The main goal of the registration protocol is to establish a mutually authenticated channel between the user and WS that allows for the distribution of a new digital identity. The security properties of this identity, that will depend on the properties of the established channel, are:

- The new identity needs to be authentic. This requires users to be somehow authenticated, and that the registration authority is also authenticated. Otherwise, impersonation or man-in-the-middle attacks might occur.
- The new identity needs to be confidential: only the user should learn it.

Step 2. Attacker capabilities and security model. As baseline, we adopt the Dolev-Yao model [93]. Thus, all communications are virtually routed through an attacker. Hence, it can automatically read the contents of every non-encrypted message. On the other hand, the cryptographic primitives are assumed to be perfect. That is, the attacker cannot obtain the contents of an encrypted message unless she somehow obtains the

appropriate key. Finally, the +/- *capabilities* list for customizing the Dolev-Yao model is just composed by one item. Namely: *"Attackers know all the information required to start a new registration process on behalf of every possible user"*. Since email addresses will be used as identifiers and as mechanisms for proving new users' identities, this means that attackers at least know the email address of a meaningful set of potential users.

Step 3. Entities assumptions, design candidate and channels abstraction.

- **Design candidate**. In this case we start from the design given in Figure 5.1. However, in general, a new design should be created at this point, or obtained as a modification of an existing one that has been proven to be insecure in a previous iteration.

- **Entities assumptions**. The entities described in Section 5.2.1 are assumed to follow the model given below.

  - **Users**. Users are assumed to be honest. In particular, they are trusted not to share the confidential tokens received from MS and WS.
  - **MS**. This is a semi-honest entity (see Definition 19). More concretely, it is trusted to correctly perform the password-based authentication of its users, and to forward email messages (in both directions) without modifying their contents. Even though this is a somehow strong assumption, it is reasonable since, in the real world, modifying or blocking email messages would be a too intrusive action that would not probably remain unnoticed by its users.
  - **WS**. The Web Server is assumed to be a honest party. Note that this is a reasonable assumption since, in particular, our protocol does not specify how to actually generate the final identities: it is intended to create a mutually authenticated channel through which the distribution of such identities is possible. In this matter, WS is interested in successfully authenticating the users registering in its service. Otherwise, its quality would undoubtedly be reduced. In contrast, it might not be realistic to assume a honest WS for the generation of identities, since it could use that information to (e.g.) impersonate users in certain situations. However, that problem is out of the scope of our protocol.

- **Communication channels**. Concerning the communication channels abstraction and the derived practical implementation, we use three types of channels in the candidate design:

  - $\mathtt{A}_S\mathtt{C}$ channel. When a message between a client and a server is sent through channels of this type, the client is certain of the server's identity. Additionally, the message is not leaked to the attacker during its transmission. In practice, this channels can be implemented with server-authenticated SSL/TLS sessions.
  - $\mathtt{A}_o$ channel. These channels ensure the authenticity of the origin/sender. However, confidentiality is not provided. As a practical approach for implementing this channels, we suggest making use of DKIM [70] as a suitable possibility.

- AC channel. The mutual authenticity property is actually achieved in two ways in the current design candidate. When the User contacts MS (steps 4-6 in Figure 5.1), a server-authenticated SSL session is established. However, when the User provides a valid username and password, she is authenticated to MS. Hence, at step 6, the channel is already mutually authenticated. In the communication between MS and WS at step 9, since both entities are servers, both are assumed to own digital certificates and trust in their corresponding CAs, so they can establish a mutually authenticated session. In both cases, the channel may be easily made confidential, by establishing an SSL session. However, we note that for the communication between MS and WS this is not required.
- P channel. That is, a channel that does not provide any security guarantee at all. Since there is no security requirement, any real channel would be appropriate. However, we make use of email messages as concrete realizations of this channels.

Step 4. informal requirements. This step is intended to provide an initial contact with the formal security requirements of each of the tokens involved in the protocol. Informally, we use the term token, or element, to refer to the minimal component of a message that may be required some security property. Every protocol step is analyzed for defining what is expected from each involved element. We call these "expectations" *informal requirements*. Table 5.1 shows the result obtained step by step (for simplicity, only confidentiality and authenticity are considered, although other properties may be needed [82]). For instance, when the $nonce'_{eml}$ value is first transmitted, at step 3, since it is sent through an insecure channel of type P (an email message), it is not required to provide any security property. This models the fact that email messages can be faked and/or eavesdropped. But, when it is compared with $nonce_{eml}$, transmitted in a server-authenticated and confidential session at step 2, then the User has the certainty that the WS initially sent the message. Similarly, the email address sent by the User at step 1 does not provide any trust guarantee, since anyone may have sent it. But when the User successfully authenticates to the MS using her password, then the email address turns into a client-authenticated token. Equivalent reasoning could be applied to other tokens. The most important one is ID. When it is transmitted, the User has already been authenticated by using the MS as "trusted sponsor" at steps 7-8, showing the correct credentials at steps 9-10. Hence, the channel used in step 11 seems to be (informally) a mutually authenticated channel. This is shown by marking up the ID token as mutually authenticated. It is also worth to be emphasized that elements that should intuitively be required to satisfy some requirement from the beginning may not be required that property until the following steps. For instance, the *ticket* element is always transmitted through SSL channels. However, since attackers may also start registration requests, it will only be considered confidential (and user-authenticated) at the final step.

Finally, we also have to decide which tool to use in the formal verification. Here, ProVerif [42] is a good candidate, since it allows the verification of the authenticity and confidentiality requirements that we have posed.

Step 5. Informal verification. We proceed to informally verify the properties assigned to the message components at step 4, considering the attacker model and the channels abstraction produced at steps 2 and 3, respectively.

Step in Figure 5.1	email	*ticket*	$nonce_{eml}$	$nonce'_{eml}$	*pwd*	ACK	$ticket'$	*ID*
1. $U \rightarrow WS : email$	none							
2. $WS \rightarrow C : ticket, nonce_{eml}$	none	A_{WS}	A_{WS}, C					
3. $WS \rightarrow MS : email, nonce'_{eml}$	none	A_{WS}	A_{WS}, C	none				
4. $U \rightarrow MS : email, pwd$	none	A_{WS}	A_{WS}, C	none	C			
5. MS : VerifyPwd	none	A_{WS}	A_{WS}, C	none	C			
6. $MS \rightarrow U : nonce'_{eml}$	A_U	A_{WS}	A_{WS}, C	none	A_U, C			
7. $U : nonce'_{eml} \stackrel{?}{=} nonce_{eml}$	A_U	A_{WS}	A_{WS}, C	none	A_U, C			
8. $U \rightarrow MS$: ACK, $nonce_{eml}$	A_U	A_{WS}	A_{WS}	A_{WS}	A_U, C	A, C		
9. $MS \rightarrow WS$: ACK,$nonce_{eml}$	A_U	A_{WS}	A_{WS}	A_{WS}	A_U, C	A		
10. $U \rightarrow WS : email, ticket'$	A_U	A_{WS}	A_{WS}	A_{WS}	A_U, C	A	none	
11. $WS : ticket' \stackrel{?}{=} ticket$	A_U	A_{WS}	A_{WS}	A_{WS}	A_U, C	A	none	
12. $WS \rightarrow U : ID$	A_U	A, C	A	A	A_U, C	A	A, C	A, C

Table 5.1: The rows under first column reference the corresponding step in Figure 5.1. A_E depicts tokens authenticated by entity E. A depicts tokens authenticated by all the entities involved in their transmission. C denotes a confidentiality requirement. *none* means that the token has already been sent, but without any assumption on its security properties. An empty cell informs that the token has not been sent yet.

Message 1. *email* has an informal requirement of *none*. This is compatible with $A_S C$channels and the attacker capability enabling her to start registration.

Message 2. *email* does not change. *ticket* and $nonce_{eml}$ must be at least WS-authenticated. We cannot yet require them to be confidential because the registration could have been initiated by an attacker. Again, since we use a $A_S C$channel, the data is WS authenticated, and no incompatibility occurs.

Message 3. The properties of each existing element are not modified. Additionally, $nonce'_{eml}$ and *email* are sent via a P channel. However, since they have *none* as informal requirement, this is not an inconsistency.

Message 4. All previous elements are unmodified. Now, *pwd* is transmitted via a $A_S C$channel. Thus, the required C property is compatible.

Check 5. *ticket*, $nonce_{eml}$ and $nonce'_{eml}$ are unmodified, but both *email* and *pwd* now have a client-authenticated informal requirement. However, at this step, the MS has verified that the provided password is associated to the specified email account. Thus, whoever provided them has been completed an authentication with the MS and it is client-authenticated. Accordingly, the associated tokens "inherit" that property, and no incompatibility occurs.

Message 6. $nonce'_{eml}$ has a requirement of *none*, which is admissible by the $A_S C$channel used to transmit it in a previous step. Note however that the channel employed in this step is a ACchannel, since the server was initially authenticated (message 4) and the user was authenticated via a username and password pair at check 5.

Check 7. If it is satisfied, $nonce'_{eml}$ inherits $nonce_{eml}$ properties.

Message 8. If check 7 succeeds, U sends back *ACK* and $nonce_{eml}$ via a ACchannel. Hence, no requirement is broken.

Message 9. Just *ACK* is modified. In this case, the confidentiality requirement is dropped from the informal requirements. Since we are still using a $\mathtt{A}_o$channel with additional confidentiality, this is not a problem.

Message 10. The *ticket*′ is sent within this message. Since we do not know from whom does it come, it does not have any informal requirement and no conflicts occur. The other elements are unchanged.

Check 11. After a successful check, the WS will have the certainty that the user with *email* email address has actually acknowledged the registration process. Hence, all tokens will be mutually authenticated.

Message 12. With all the components authenticated, the WS generates the identity ID and sends it through a ACchannel. Therefore, the authenticity and confidentiality properties of the ID are kept. Additionally, we can now require *ticket* to be confidential, since ID is only issued for verified users and, in that case, the initial SSL session should have been confidential to the attacker.

We remark the special care taken in matching the informal requirements expected from the transmitted tokens with the properties of the communication channels. Satisfied the informal verification, we proceed to the formalization of the protocol. Moreover, with the insight gained after this preliminary analysis, formalizing the protocol will be easier. Additionally, independently of the second phase, at this point we have already reached the PAL1 assurance level defined in [142] (which includes a semiformal protocol specification, informal security and attacker models, informal security requirements and informal proofs).

5.3.2 Procedural verification of security

Having performed an informal evaluation, we apply the steps of the formal verification phase of the methodology in Chapter 3 as follows.

Step 6. Protocol pseudocode. From the design candidate obtained in the informal phase, and shown in Figure 5.1, we create a more detailed description in the shape of pseudocode. In this case, the tool we chose for this formal verification is ProVerif, since it allows to check all the required properties. Therefore, we make this pseudocode resemble the final program for easing the subsequent coding task. The produced pseudocode is available online[7].

Step 7. Formalization of the protocol. Using the pseudocode as starting point, we produce the final formalization for the chosen tool/method (ProVerif in our case). The details are given below, in terms of the communication channels and participating entities. However, for the sake of readability, instead of explaining the three entities, we center our attention on the User and MS processes and just make a few comments on the WS logic. Certainly, given the details of User and MS, it is easy to grasp the behavior of the WS. Besides this analysis, the associated source code, ready to be processed with ProVerif, is publicly available online[8]. We summarize the notation employed in the source code included in the listings below in Table 5.2. This notation includes both native ProVerif instructions and naming conventions used in our code.

[7] http://www.ii.uam.es/~gnb/thesis-jdv.tgz
[8] http://www.ii.uam.es/~gnb/thesis-jdv.tgz

Element / Instruction	Description
$msgX$	Label for the X-th message sent in a legitimate protocol run
$kssluwsX$	X-th SSL key negotiated between U and WS
$ksslumsX$	X-th SSL key negotiated between U and MS
`msaccounts`(*email*,*pwd*)	Database managed by MS, containing pairs (*email*,*pwd*)

Table 5.2: Specific notation used in the ProVerif code listings for SEBIA besides the one defined in Table 2.1. It includes both native ProVerif instructions and naming conventions adopted in this chapter.

Communication channels. As stated, our protocol makes use of three types of communication channels. The details about how does our model simulates them is given next.

- $\mathtt{A}_S\mathtt{C}$**channel**. The abstraction employed for this channel must ensure the following:

  1. The transmitted information is kept secret.
  2. The client must be certain of the server's identity.
  3. The server may not be certain of the client's identity.

  We first instantiate the private channels `privateSSLuserchannel`, `privateSSLwschannel`, `privateSSLserverchannel`, and the process `sslkeynegotiation`. This process will be waiting for an incoming message via `privateSSLuserchannel`, which is only used by the user for sending messages. Moreover, since the channel is declared as private, the attacker does not have access to it or the messages sent through it. When the process receives a message containing a new nonce (for avoiding mixing different sessions), it will create a key for symmetric encryption. This key is sent back to the client via the channel `privateSSLuserchannel` and to the server (who will be awaiting for it in order for the process to be activated) via `privateSSLmschannel` or `privateSSLwschannel` (depending on whether the server is MS or WS, respectively). Since the channels used to distribute the freshly created key are private, the attacker does not learn the keys. Since the keys are private, they can be used to encrypt and send messages over any publicly accessible channel, which will be kept secret.

  However, this actually provides mutual authentication, since the attacker cannot make use of the employed private channels and thus cannot be the originator of any message encrypted with the created keys. Since we need just server-authenticated channels, we create a process equivalent to `sslkeynegotiation`, named `sslbypass` in which the private channel `privateSSLuserchannel` is substituted by a public channel. Therefore, the attacker can now initiate key exchanges, and the server will still receive the created keys via `privateSSLmschannel` (or `privateSSLwschannel`).

  Finally, note that this is the precise definition of a SSL/TLS channel in which the symmetric keys have been negotiated with server-authentication. Thus, it could be easily deployed in practice. The code for the sslkeynegotiation process is depicted in Figure 5.2 (the code for the *bypass* is omitted, since it is mostly the same).

```
1  (** SSL negotiation process: used only between User and WS **)
2  let sslkeynegotiation =
3
4
5          in(privateSSLuserchannel, (u:Host,s:Host,n:Nonce));
6
7          if s = ws then (
8                  new kssluws:Key;
9                  out(privateSSLuserchannel, (ws,u,n,kssluws));
10                 out(privateSSLwschannel, (u,ws,n,kssluws))
11
12         )
13
14         else (
15                 if s = ms then (
16                         new ksslums:Key;
17                         out(privateSSLuserchannel, (ms,u,n,ksslums));
18                         out(privateSSLmschannel, (u,ms,n,ksslums))
19                 )
20         ).
```

Figure 5.2: Process for establishing SSL sessions. A message is received from the `privateSSLuserchannel`, if it is aimed to WS (`s = WS`), a new SSL key shared between U and WS (`kssluws`) is created; else, the same process occurs for creating SSL keys shared between U and MS. Since `privateSSLuserchannel` is private, the attacker cannot read from/write into it.

- $\mathtt{A}_o$**channel**. This channel is modeled by generating public-private keypairs for the originating entity, and distributing the public part securely to the recipient. In the code, the originating entity will be MS, and the recipient will be WS. For the secure distribution, the WS receives the public key as a parameter. This is necessary since otherwise, the attacker could just generate its own keypair and issue signed messages. In this manner, any message signed by MS will be origin authenticated and just sending it via a public channel will actually model an origin-authenticated channel.

- `P`**channel**. Modeling insecure channels does not require any special consideration other than declaring the channels as public (more precisely, not declaring them as private).

The User process. This process represents users willing to be registered in a new site. In our model the user has three attributes: a hostname, an email account, and a password for that email account. They are represented in the code in Figure 5.3 through Figure 5.7 as `u`,`e` and `pwd`, respectively. The hostname `u` represents the "name" or IP address of the machine of the user. It is not a field strictly necessary for our protocol, but it helps to determine the messages source. Note that it is declared as a new name and always sent encrypted, but we make it public by sending it over the public channel `net` in order to model that the attacker might know the hostname (or IP address) of its victims. The email `e` is the actual identifying information used by our protocol, and it is also used to simulate the fact that each specific WS may restrict the admissible email domains. Thus, legitimate users (i.e., users with valid emails) declare their email as a

new name within their code, and insert it into a table named `emails` used by the WS to check that it is a valid email address. The email is also inserted along with the associated password `pwd` in a table named `msaccounts` used by the MS to authenticate its email users. Since tables are not accessible to the attacker in ProVerif, this successfully models our assumption that only the legitimate owner of an email account can prove ownership to the MS. Additionally, the email address is immediately sent out via the public network to emulate the very possible fact that an attacker may know the email address of its victim. The code associated to all these initializations is shown in Figure 5.3.

```
1  let userprocess =
2
3          new u:Host; out(net,u); (* The machine name *)
4          new e:Email; insert emails(e); out(net,e);
5          new pwd:Password; insert msaccounts(e,pwd);
```

Figure 5.3: User process code: initialization.

Once all the user's internal data has been set, the user can start an SSL/TLS session by sending out a message through the `privateuserchannel` (in line 3 of Figure 5.4). The first field of this message specifies the host initiating the session, the second field determines the other end of the communication (`ws` comes from Web Server), and the third field is a *nonce* to avoid mixing up keys generated for different SSL sessions (this is possible due to a subtlety in ProVerif, that allows the reasoning engine to resend messages sent via private channels). Once the SSL session has been established, the user sends to the WS her email address via SSL in the the message in line 8 of Figure 5.4. Here, the second and third fields determine the sender and receiver, the last is the email, and the first is just a tag to ease debugging the protocol (the number indicates the order in which these messages are sent). In the last step of this SSL session (lines 11-13 of Figure 5.4) the user receives the ticket generated by the WS, along with the nonce that will allow the user to verify the email that the WS has already sent at this point. Only messages with the expected format, and encrypted using the SSL key negotiated before will be accepted at this point. Since the key has been negotiated in with server authentication, the user has guarantees that it indeed comes from the WS.

```
1          (* Establish the first SSL session *)
2          new n1:Nonce;
3          out(privateSSLuserchannel, (u,ws,n1));
4          in(privateSSLuserchannel, (=ws,=u,=n1,kssluws1:Key));
5
6          (* Send the registration request *)
7          event UserReqReg(e);
8          out(net, senc((msg1, u, ws, e), kssluws1));
9
10         (* Receive via SSL the SSL ticket and the email nonce *)
11         in(net, cmsg3:bitstring);
12         let (=msg3, =ws, =u, =e, ticketssl:Ticket, ne:Nonce) =
13             sdec(cmsg3,kssluws1) in
```

Figure 5.4: User process code: Initial registration request.

Once the User receives the SSL ticket and the email nonce, she waits for an email from the WS. Thus, she establishes an SSL connection with her MS and gains access through authenticating herself by means of email address and the associated password (line 10 in Figure 5.5). If the MS has received the email, it sends it back to the user, also encrypted via the same SSL session. The user receives and decrypts it (lines 11 and 15, respectively) and in case the *nonce* contained within the email is the same than the *nonce* received via SSL (this is specified with the `=ne` syntax in line 15), then she sends to the MS an instruction to send an authenticated `ACK` message back to the WS. This last step is performed in line 18 of Figure 5.5.

```
1        (* Fetch from the MS the email with the ticket sent by the WS *)
2
3        (* First: establish an SSL session with MS *)
4        new n2:Nonce;
5        out(privateSSLuserchannel, (u,ms,n2));
6        in(privateSSLuserchannel, (=ms,=u,=n2,ksslums1:Key));
7
8        (* Second: Authenticate and receive the email *)
9        event UserAuthMS(e,pwd);
10       out(net, senc((msg4,u,ms,e,pwd),ksslums1));
11       in(net, cmsg5:bitstring);
12
13       (* If the nonce received by email matches the one received
14          via SSL, instruct the MS to send an ACK *)
15       let (=msg5, =ms, =u, =ne) = sdec(cmsg5,ksslums1) in
16
17       event UserReqSendACK(e,ticketssl,ne);
18       out(net, senc((msg6,u,ms,e,ne,pwd),ksslums1));
```

Figure 5.5: User process code: Secure email verification and acknowledgment.

In the last step the user has to send back the SSL token to the WS in order to complete the registration. To do so, the user establishes a new SSL session in lines 3 and 4 of Figure 5.6. Indeed, this models the probable fact that the user could need to establish a new SSL session with the WS instead of using the previous one (e.g., the user can perform this last step the next day after initiating the registration, or after rebooting her machine). Finally, the user sends the SSL ticket back to the WS (in line 6). When the WS receives the ticket, it will check that it matches the ticket sent to this user and that the `ACK` sent by the MS has already been received. This last check is of the utmost importance in order to be sure that the user is the owner of the email address. Subsequently, the identity is generated and sent to the user, who receives and decrypts it in lines 7 and 9 of Figure 5.6. Again, given that the SSL session has been established with server authentication, the user can correctly assume that the new identity actually comes from the WS.

Therefore, Figure 5.3 through Figure 5.6 compose a realistic model of the typical users that would take part in our registration protocol.

The Mail Server. The MS process emulates the behavior of the mail servers that take part in our protocol. The initialization of this process just consists of receiving two parameters: the MS private key, and the WS public key. Both keys are generated in the "main" process that initializes the environment and launches the processes corre-

```
1        (* Send both tickets back to the WS in a new SSL session *)
2        new n3:Nonce;
3        out(privateSSLuserchannel, (u,ws,n3));
4        in(privateSSLuserchannel, (=ws,=u,=n3,kssluws2:Key));
5
6        out(net, senc((msg8,u,ws,e,ticketssl),kssluws2));
7        in(net, cmsg9:bitstring);
8
9        let (=msg9,=ws,=u,=e,=ticketssl,id:Id) = sdec(cmsg9,kssluws2) in
10       event UserRecvId(e,id,ticketssl,ne);
11       0.
```

Figure 5.6: User process code: Identity retrieval.

sponding to each entity. Indeed, this is a reasonable way of modeling the *real world* scenario in which each trusted server has its own private key, certified by some Certification Authority, and the public key is accessible to anyone else. The MS is "invoked" by receiving a message containing a nonce, and having as final destination a certain email address. This is performed in line 5 of Figure 5.7. Note that it is sent via the public channel (`net`) without neither encryption nor digital signatures, which models the real scenario where the MS cannot determine the email source (it could be either the WS or an attacker) and eavesdropping is possible (since an attacker can read the email sent by the WS).

After receiving the message with the nonce, the MS waits until a new SSL session is established and someone provides a valid email account along with the associated password (checked by accessing the `msaccounts` table). This is done in lines 11-15 in Figure 5.7. When that occurs, the MS sends to the user (line 20 in Figure 5.7), and through the recently established SSL session, the email previously received. Hence, at this point the legitimate owner of the email account has received the registration email (lines 11-15 in Figure 5.5). When the MS receives from the user (lines 24-25 of Figure 5.7) the message instructing it to send the `ACK` message to the WS, prepares it and sends it along with the corresponding digital signature (in line 29).

The Web Server. For readability, we omit the WS code, since its behavior is determined by the code presented for the User and the MS (the model source code is completely available online[9]. However, it is worth noting that the WS is charge of a set of verifications that should be carried out very carefully. First, it needs to verify that the email address belongs to a trusted domain (and that this domain provides the `ACK` functionality). After sending the SSL ticket and the email nonce via the SSL session, and the email to the MS, the WS will mark the status of the current registration as *mail verification pending*. It is of the utmost importance that the WS does not continue the registration process until it receives the digitally signed `ACK` message from the MS. After receiving the `ACK` message and the SSL ticket (and verifying that the ticket matches the one sent during this registration), the WS has the certainty that whoever started the registration is the owner of the email account. Thus, it can complete the registration process by creating and sending the digital identity using the same SSL session established to send back the SSL ticket.

[9] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

```
1  let msprocess(kmsprv:PrvKey, kwspub:PubKey) =
2
3          (* Receives an email *)
4          (* Note that it is not encrypted nor authenticated *)
5          in(net, (=msg2, =ws, =ms, e:Email, ne:Nonce));
6
7          (* Receives a SSL session *)
8          in(privateSSLmschannel, (u:Host,=ms,n2:Nonce,ksslums1:Key));
9
10         (* Receive authentication via the SSL session *)
11         in(net, cmsg4:bitstring);
12         let (=msg4,=u,=ms,=e,pwd:Password) = sdec(cmsg4, ksslums1) in
13
14         (* Check the password *)
15         get msaccounts(=e,=pwd) in
16
17         event MSVerifAcc(e,pwd);
18
19         (* Send email securely via SSL *)
20         (* Note that at this point, the user is authenticated to MS *)
21         out(net, senc((msg5,ms,u,ne), ksslums1));
22
23         (* Wait for an instruction to send ACK *)
24         in(net, cmsg6:bitstring);
25         let (=msg6,=u,=ms,=e,=ne,=pwd) = sdec(cmsg6,ksslums1) in
26
27         (* Send signed ACK to WS *)
28         event MSSendsACK(e,ne);
29         out(net, ((msg7,ms,ws,e,ne), sign((msg7,ms,ws,e,ne),kmsprv)));
30         0.
```

Figure 5.7: Mail Server process code.

Step 8. Procedural verification. With the protocol formalization, all that is left is to produce the formal security "queries" and evaluate them with the chosen tool/method. This is done with the help of Algorithm 3.4 in Chapter 3, which is based on the informal requirements produced at step 4. Basically, the algorithm goes through all the informal requirement, starting at the last protocol step, until all the elements are assigned some requirement (or *none*) or all the messages/checks of the protocol are explored. In this case, the result is straightforward, since all the elements are assigned a requirement at the final step. We now specify the final requirements for each element, after applying the mentioned algorithm, and how they are verified with the code along with the result produced by ProVerif. The results here shown can be reproduced with the code available online[10]. Note that the *ticket*′ and *nonce*′ elements are not explicitly included in this analysis, since after a successful execution, they must be the same as the original ones. Hence, if the protocol is secure for the latter, it is secure for the former too.

- *email* (*Requirements*: User Authenticity). This is verified in ProVerif using the events `UserAuthMS(email,pwd)` and `MSVerifiesAccount(email,pwd)`. Events in ProVerif are operations that can only be executed by legitimate processes, by definition. Correspondence chains of these events, with variables, are used to prove authenticity [42]. They can be injective or not injective. For simplicity, we do not make this dis-

[10] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

tinction here, even though it is made in the provided source code. The former is executed only by users and the latter only by the MS. Then, we ask ProVerif whether or not the latter is always preceded with at least one of the former, to which ProVerif answers with:

```
RESULT event(MSVerifAcc(email,pwd))
  ==> event(UserAuthMS(email,pwd)) is true.
```

- *ticket* (*Requirements*: Authenticity, Confidentiality). To prove user-side authenticity, we use the events `UserReqReg(email)` and `WSSendsTickets(email,ticket, nonce)`. Specifically, the latter, invoked by the WS, must be preceded by the former, invoked by the User. For server-side authenticity, we use the event `UserReceivesID(email,id, ticket,nonce)`, executed by the User, which must be preceded by `WSSendsTickets(email, ticket,nonce)` and `UserReqReg(email)`, executed by the WS and User, respectively. To both queries, ProVerif returns:

```
RESULT event(UserRecvId(email,id,ticket,nonce))
  ==> event(WSSendsId(email,id,ticket,nonce)) is true.

RESULT event(UserRecvId(email,id,ticket,nonce))
  ==> (event(WSSendsTickets(email,ticket,nonce)) &&
       event(UserReqReg(email))) is true.
```

  For confidentiality, we use a workaround. Since attackers can initiate registration requests (base assumption), they can actually receive *ticket* tokens, even though these tickets may not be finally used to obtain an ID. However, since ProVerif explores the execution flow in order, once the attacker obtains a ticket, it will output that the requirement is violated. To model this, we use an additional ticket, named `acctick-etssl` , which is only created when a ticket is accepted by the WS. This ticket is also sent to the public network, but encrypted with the SSL key negotiated for the last message exchange. Hence, if the attacker gains this token, it is because she has successfully completed all the previous steps. Asking ProVerif about this, it outputs:

```
RESULT not attacker(acctickettssl) is true.
```

- $nonce_{eml}$ (*Requirements*: Authenticity). The reasoning made for proving authenticity of *ticket* holds here, since $nonce_{eml}$ is covered by the same events.

- *pwd* (*Requirements*: User Authenticity, Confidentiality). Confidentiality is proved with: `query pwd:Pwd; attacker(new pwd)`, to which ProVerif answers:

```
RESULT not attacker(pwd) is true
```

  For authenticity, the event correspondences used to prove *email* authenticity are valid, since they also include the password.

- `ACK` (*Requirements*: Authenticity). This requirement is demonstrated with three different events, `event(WSProcACK(email,ticket,nonce))`, `event(MSSendsACK(email,nonce)` and `event (UserReqSendACK(e,ts,n))`. The parameters are the ticket and nonce sent

because they are actually what define the ACK uniqueness in the real protocol. The first event is executed by the WS, the second by the MS and the latter by the User. Finally, the WS event must be preceded by the other two. ProVerif responds to this query that the correspondence is maintained:

```
RESULT event(WSProcACK(email,ticket,nonce))
  ==> (event(MSSendsACK(email,nonce)) &&
       event(UserReqSendACK(email,ticket,nonce))) is true.
```

- *ID* (*Requirements*: Authenticity, Confidentiality). Authenticity is proved with the events UserRecvId and WSSendsId already used for *ticket* authenticity. Confidentiality is verified with: `query id:Id; attacker(new id)`, to which ProVerif outputs:

```
RESULT not attacker(id) is true.
```

Finally, having demonstrated that all the security requirements are kept, we can safely conclude that the model of our design candidate is actually secure. Still, in the next section we provide a supplementary analysis, showing the necessity of all the elements included in our approach.

5.3.3 A minimality analysis

In this section we study why are the main components of the protocol necessary. This includes the SSL/TLS ticket, the ACK message and the email nonce. However, we do not include here the general purpose message components like the message tags, which are used for easing the debugging process, nor the sender and recipient identities, which are a general good practice in protocol design, as stated in [5]. Additionally, we will finish this section by making some remarks on instruction *ordering* issues that might thwart the protocol's objective, lest they are executed in the wrong order. We will use the notation $enc_K(\cdot)$ when a message is sent (symmetrically) encrypted under the (symmetric) key K, and $sign_{prv(H)}(\cdot)$ when a message is sent signed under the private key of host H. For readability, we do not include the SSL/TLS key negotiations, and just use the name K_{SSL_i} for the key negotiated and used during the i-th SSL session within an attack trace. We also omit in the attack traces shown below the message tags and the sender and recipient identifiers that were included in the messages of the formal definition of the protocol.

SSL/TLS ticket. This element plays a key role in our protocol, since it allows us to guarantee that the user who starts a registration request will be the only one capable to complete it. This can be tested by removing the field ticketssl from all the messages (and events) in the source code available online[11]. Once this so weakened model of the protocol is run with ProVerif, the (naive) attack trace informally shown in Figure 5.8 is returned.

ACK message. As we have emphasized above, the ACK message sent to the WS is essential in our protocol. It is the message guaranteeing that the User is indeed the owner of the email address that she specified. Without it, even with the SSL ticket, an

[11] http://www.ii.uam.es/~gnb/thesis-jdv.tgz

$User \rightarrow WS : enc_{K_{SSL_1}}(email)$
$WS \rightarrow User : enc_{K_{SSL_1}}(Nonce_{email})$
$WS \rightarrow MS : Nonce_{email}$
$User \rightarrow MS : enc_{K_{SSL_2}}(email, password)$
$MS \rightarrow User : enc_{K_{SSL_2}}(Nonce'_{email})$
User: Checks $Nonce'_{email} = Nonce_{email}$
$User \rightarrow MS : enc_{K_{SSL_2}}(ACK)$
$MS \rightarrow WS : sign_{prv(MS)}(ACK, Nonce_{email})$
$Attacker \rightarrow WS : enc_{K_{SSL_3}}(email)$
$WS \rightarrow Attacker : enc_{K_{SSL_3}}(ID(email))$

Figure 5.8: Attack trace when the SSL ticket is not used. The attacker might wait until a legitimate user starts a registration and instructs the MS to send the ACK message. Afterwards, blocking the user's communications and establishing an "illegitimate" SSL session with the WS she would be able to retrieve the user's identity.

attacker would be able to obtain an identity related to any user. Again, removing the ACK message from the provided code[12],we obtain the attack trace shown in Figure 5.9 with ProVerif.

$Attacker \rightarrow WS : enc_{K_{SSL_1}}(email_{victim})$
$WS \rightarrow Attacker : enc_{K_{SSL_1}}(Ticket_{SSL}, Nonce_{email_{victim}})$
$WS \rightarrow MS : Nonce_{email_{victim}}$
$Attacker \rightarrow WS : enc_{K_{SSL_2}}(Ticket_{SSL}, email_{victim})$
$WS \rightarrow Attacker : enc_{K_{SSL_2}}(ID(email_{victim}))$

Figure 5.9: Attack trace when the ACK is removed from the protocol. The attacker starts from the beginning a new registration process, specifying the email address of its victim.

In our protocol, the email nonce is used to guarantee that the ACK received corresponds to a single email sent to the user, and to allow the user to check that the email received actually corresponds to the original request. Hence, it might be tempting to also use it to prove that the user has read the email (this is exactly what is done in classic EBIA). However, given that the email is sent unencrypted, an attacker could easily obtain it and send it back to the WS in the last SSL session. Thus, just using the knowledge of the email nonce as an *implicit ACK* is not enough. Indeed, this is the assumption in classic EBIA that makes the protocol insecure.

Email nonce. If the WS does not include the email nonce in the same message where the SSL ticket is sent, another attack is enabled, since the user is not able to link her legitimate registration request with the received registration email. Thus, even by sending the ACK message, an attack is possible. By removing the email nonce from message 3 in Figure 5.1 (and thus the check in message 5 that compares the nonce received via SSL and the nonce in the email), we encounter the attack trace depicted in Figure 5.10, after some polishing of the trace produced by ProVerif.

[12] http://www.ii.uam.es/~gnb/thesis-jdv.tgz

$User \rightarrow WS : enc_{K_{SSL_1}}(email)$
$WS \rightarrow User : enc_{K_{SSL_1}}(Ticket_{SSL})$
$WS \rightarrow MS : Nonce_{email}$
$Attacker \rightarrow WS : enc_{K_{SSL_2}}(email)$
$WS \rightarrow Attacker : enc_{K_{SSL_2}}(Ticket'_{SSL})$
$WS \rightarrow MS : Nonce'_{email}$
$User \rightarrow MS : enc_{K_{SSL_3}}(email, password)$
$MS \rightarrow User : enc_{K_{SSL_3}}(Nonce'_{email})$
$User \rightarrow MS : enc_{K_{SSL_3}}(ACK(Nonce'_{email}))$
$MS \rightarrow WS : sign_{prv(MS)}(ACK, Nonce'_{email})$
$Attacker \rightarrow WS : enc_{K_{SSL_4}}(Ticket'_{SSL}, email)$
$WS \rightarrow Attacker : enc_{K_{SSL_4}}(ID(email))$

Figure 5.10: Attack trace when the email nonce is not sent jointly with the SSL ticket. The attacker can make the WS believe that it is issuing an ID related to an acknowledged email account, to the same user who requested to be registered.

Note that the WS receives two registration requests related to the same email. However, this may very well be a normal situation. For instance, a user could request registration but, somehow, lose the received SSL ticket and thus, in order to complete registration, she would need to fetch a new one. Taking advantage of this, when the attacker sees its victim requesting registration, she sends a new request for the same email address, and gets a valid ticket. The user may receive two registration emails. However, she could easily think that it is a server malfunction (after all, she has actually requested registration). Furthermore, the attacker could even block one of those two emails, in order to avoid suspicion. After the user sends the `ACK`, the attacker establishes a new SSL session with the WS and provides it the ticket obtained in the previous step. Since the WS has already received the `ACK`, it sends the identity to the attacker.

5.3.4 The importance of a correct ordering

From the previous subsections we can already deduce that the checks performed by the users and the WS are of the greatest importance. However, it is worth emphasizing that not only the checks, but also the order in which they are performed with respect to the messages reception and sending, is crucial for the protocol.

Firstly, the user must check that the nonce contained in the received email actually matches the nonce received altogether with the SSL ticket. Once this check is successfully performed, and only then, the user can safely instruct the MS to send the signed `ACK` to the WS. If this order is not followed, an attack similar to the one shown in Figure 5.10 is enabled.

Secondly, the WS needs to wait until receiving the signed `ACK` message from the MS before completing a registration request. That is, even though the WS receives back the SSL ticket from an already initiated registration request, it must not finalize it until the `ACK` has been received. As a matter of fact, the `ACK` is the only token proving that the user who is sending back the SSL ticket is the owner of the email address. If this order is broken, an attack like the one depicted in Figure 5.9 would be possible.

5.4 Usability, additional costs and trust assumptions

As we have shown, our protocol guarantees authenticity of the registering users and secrecy of the created identities. Thus, it implements a correct registration protocol. Still, it remains to analyze how usable our protocol is, and whether the introduced modifications imply acceptable costs (both in terms of communication/computational costs and in terms of the required modifications on the underlying infrastructure) or if they are otherwise unrealistic.

From the design complexity perspective, and concerning the necessary modifications to the underlying infrastructure, our protocol strictly follows the properties of the infrastructure that is currently being used by EBIA systems. Namely, we have considered the security properties provided by the available communication channels, i.e., confidential and server authenticated SSL/TLS sessions, and unencrypted and unauthenticated email messages. The only necessary addition is an authenticated channel between two trusted servers (the Mail Server and the Web Server performing the registration). However, even though this supposes an extension to the typical functionality of mail servers, it is not a too cumbersome functionality to be implemented. In fact, the *DomainKeys Identified Mail* (*DKIM*) [70] method provides a very close functionality. By means of DKIM, a signing domain digitally signs an email message, gaining some responsibility over it. Many email providers (e.g. Gmail, Yahoo) use it to reduce spam by signaling *trusted* emails Also, according to `http://eggert.org/meter/dkim`, it is a widely deployed technology. For instance, an email provider could easily implement our `ACK` method by adding a few email filtering rules in conjunction with DKIM[13].

From the point of view of the communication costs, we show in Table 5.3 a relation of which messages are added with respect to EBIA and with respect to the equivalent solution of [182]. We also compare SEBIA with [182] because it is the one that most resembles our proposal in terms of aims and construction. Note that three messages are added when compared to EBIA, and only two when comparing with [182]. With respect to EBIA, we add the SSL ticket, the nonce sent in the second message of our protocol, and the messages used as explicit acknowledgment (originated by the user and forwarded by the Mail Server). In [182] the message containing the SSL ticket is already sent, so the only addition is the set of messages involved in the acknowledgment. Moreover, each of these extra messages would probably be just a few tens of bytes, hence they do not imply an unacceptable communication over-cost. As for computational costs, compared to EBIA, our scheme adds the computation of an extra random element (the SSL ticket) by the Web Server, the creation of a digital signature by the Mail Server (the `ACK`), and two bitstring comparisons (steps 6 and 10 in Figure Figure 5.1); compared to [182], our proposal just adds the computation of the digital signature and the bitstring comparison in step 6 of Figure Figure 5.1. The bitstring comparisons are cheap operations, and there are very efficient implementations of (pseudo) random number generation and digital signatures. Besides, considering that registrations are not the most frequent action in web systems, the extra costs would probably be quite acceptable.

It is also important for the feasibility of our protocol to ponder if the imposed trust relationships are realistic. And again, they are quite bearable. Note that we are using email addresses as primary identifiers for the users that will be registered with our

[13] Although special care should be taken in some intrinsic properties of DKIM, like the possibility to replay messages, see `https://wordtothewise.com/2014/05/dkim-replay-attacks/` (accessed on March 31st, 2015).

Scheme	1	2	3	4	5	7	8	9	11
EBIA	YES	NO	YES	YES	YES	NO	NO	YES[a]	YES
[182]	YES	YES	YES	YES	YES	NO	NO	YES	YES

[a] In EBIA, $nonce_{email}$ is sent here.

Table 5.3: Relations of messages SEBIA and whether or not they are also present in EBIA or [182]. The additional messages, if absent, enable the attacks explained in previous sections. The numbers shown in the first row represent the associated message of our protocol as shown in Figure 5.1 (e.g., message 2 corresponds to the message where $ticket_{SSL}$ and $nonce_{email}$ are sent). In the column below each number, we write "NO" when the corresponding message is not present in EBIA/[182] and "YES" if it is indeed present. Note that messages 3-5 actually belong to the process of sending an email and fetching it from the mail server, hence, they are always present.

protocol. Hence, we need to trust that, when an account is compromised, the user will take the necessary measures to, at least, inform of this fact. Therefore, this trust relation depends directly on the measures that the email provider and its users take to protect accesses to their email accounts. Luckily, it is a common practice nowadays to protect authentication to mail servers using SSL/TLS and other additional security procedures, like the Google Authenticator [110]. Note also that our protocol could be extended with additional multichannel techniques during registration, like SMS or push notifications[14], to enable verification of mobile phone numbers (although guaranteeing that the owner of the specified mobile phone number and email address are the same might be more challenging than what it seems at first sight). Concerning the servers in our protocol (i.e., the Mail Server and the Web Server), they are assumed to be trusted entities which have publicly trusted digital identities. Thus, establishing mutually authenticated sessions between them, and server-side authenticated sessions with any other entity, is something trivial. Of course, we do not take into account attacks in which the user ignores server side certificates, or is deceived by seemingly legitimate, but illegitimate, ones. Therefore, it seems that all the requirements placed by our protocol, both related to technical and trust matters, are reasonable given current infrastructures.

Finally, from the usability perspective, everything could be done just like with classic EBIA. Following the approach taken in [182], a plugin for email clients could be developed to perform all the user-side tasks automatically. That is, the plugin will receive and store the SSL ticket and the expected email nonce. Once the corresponding email has been received, it would check that the nonce included in the incoming email actually matches the nonce received via SSL/TLS. Afterwards, the instructs the Mail Server to send the authenticated `ACK` to the Web Server. As a matter of fact, a preliminary version of this protocol was actually implemented and applied to the Moodle platform [77] (this preliminary version did not include the explicit authentication message although, as we have observed, this could be done without affecting usability). Moreover, we conducted a survey on its usability and concluded that it was an acceptably usable proposal (see [80, Section 5.5] for a summary of the usability test). Hence, our proposal achieves a high level of security suitable for many situations without eroding usability.

[14] E.g., Google Cloud Messaging or Apple Push Notification Service.

5.5 Distributing [anonymous] credentials with SEBIA

As we have seen, the proposed protocol allows to verify the identity of any user who owns an email account in a trusted email provider. Note also that we have not specified the concrete form of this identity. In fact, what the protocol does is to establish a mutually authenticated channel between the user and WS, having the latter a reasonable certainty that the user is the owner of the email address.

Once established a mutually authenticated channel between server and user, it is possible to run any desired method for generating personal credentials. This credentials may be a new username and password pair, a private key and the corresponding X.509 certificate, or whichever any other type of credential. The protocols employed for creating credentials many times involve additional message exchanges, for ensuring that the server does not learn any privileged information. Note however that this is not a problem since, as stated, any arbitrary number of additional exchanges may be done after the authenticated channel has been established.

Specifically, and for the case that will occupy us in the following chapters, anonymous X.509 identities may be generated in the last step of SEBIA. This anonymous digital identities may then be managed with the mechanisms defined in Chapter 6, and directly employed in the systems proposed in Chapter 7 and Chapter 8. As for their implementation, `libgroupsig` of Chapter 4 would make it much simpler.

5.6 Chapter conclusion

In this chapter, we have presented a registration protocol based on the EBIA approach for generating identities associated to email addresses. Furthermore, the security properties expected from it have been analyzed in detail using the methodology defined in Chapter 3 where, in this case, we have employed the automated verifier ProVerif.

Two aspects are worth to be noted about the proposed protocol. First, it provides security guarantees that are reasonable for many situations arising in current Internet-based systems. Moreover, it does so without eroding usability and it is deployable by means of already existing technologies. Secondly, it does not impose any requirement on the structure or contents of the generated identities. Therefore, it is suitable for any system independently on whether it uses X.509-like identities, GPG keys, username and password pairs, etc.

As for possible further work, it would be determining to actually implement it using the suggested technologies and obtain actual measurements of the associated costs. In this subject, a prototype of a variant of this protocol, based on (simulated) SMS instead of email addresses as authenticators for the users, has already been implemented using the Android platform as base system [73] (which certainly shows promising results). Note that, even though in the definition of the protocol we have used email addresses for identifying users due to the strong relation with EBIA, mobile phones numbers are also suitable for this task. Since SMS messages may be considered to provide roughly the same security as email messages [97], exchanging one for another is possible. The additional requirement would be that the telephony company also implemented a service similar to DKIM. Also, usable solutions including multiple-factor authentication methods, like Pico[15] [177] would certainly provide enhanced security. Finally, note that we place complete trust in the Mail Server (MS) entity: if it is cor-

[15] `www.mypico.org`. Accessed on December 27th, 2014.

rupted or somehow compromised, the security of the whole process cannot be guaranteed. Therefore, achieving a variant of this protocol in which this trust is reduced will certainly imply a great improvement.

As was mentioned in Section 5.5, the protocol in this proposal allows the distribution of any kind of identity, where the identifying element (or the *Subject ID* in X.509 jargon [185] would be the verified email address) via the secure channel established at the end of the protocol. Specifically, it would be possible to distribute anonymous credentials like the ones used in the following chapters of this thesis. Note that implementing this protocol for the distribution of anonymous credentials we are one step further into the process of creating privacy respectful, practical and realistic solutions, using the existing infrastructure and widely deployed technologies. Furthermore, as we will discuss in Chapter 7, this a variant of this protocol could be used in the context of privacy respectful e-commerce therein proposed.

CHAPTER 6

X.509-based fair anonymity management mechanisms

Chapter based on and supported by references [21, 78, 84].

Since its creation, the Internet has increased enormously both in volume of users and number of services provided through it. This, jointly with the increase in the computing and storage capabilities, has led to vast amounts of data being constantly processed and stored by private companies and government agencies. As a consequence, a claim for privacy has also begun to gain strength. However, while robust cryptographic techniques have been successfully applied in real systems to enhance security properties such as confidentiality and authenticity, their application to anonymity (as a mean to endorse privacy) has not been so widespread, nor the adoption of the proposed solutions so common, considering the proportion of users of any kind of anonymizing solution over the total number of Internet users. This, in spite of the fact that robust cryptographic primitives for anonymity and privacy protection have been proposed. For instance, as shown in Figure 6.1, cryptographic primitives such as, *group signatures* [65] and *blind signatures* [64] operate at the information level in order to ensure (information) anonymity; or, at the communications level, we find anonymizing networks such as *onion networks* [107] and *mix networks* [63] which ensure anonymity of the communications themselves. Among the latter, Tor [91] is worth to be noted due to its popularity. It is shipped as an standalone executable that does not hinder usability at all (besides the unavoidable perception of lower download/upload speeds).

We argue that this lack of success of anonymity based systems is based on two main facts. First, because anonymity is a double-edged sword, which improves the overall privacy of the Internet users, but that may be employed by dishonest users as a means to perform illegitimate actions. For instance, in [145] the authors run a Tor exit node for gathering statistical evidence for a further analysis of Tor's traffic. They conclude stating that it is not uncommon to see *"hacking attempts, allegations of copyright infringements, and bot network control channels"* routed through Tor. Or the study in [35], where the authors find evidence of substantial traffic generated through Tor hidden services and related to the botnet Skynet[1]. Additional informal evidence supporting the claim that the anonymity provided by anonymizing services can be (and is being) misused is the fact that there already exist services, like BlockScript[2], which include explicit func-

[1] `https://community.rapid7.com/community/infosec/blog/2012/12/06` (accessed September 8th, 2014).

[2] `http://www.blockscript.com/features.php` (accessed September 8th, 2014).

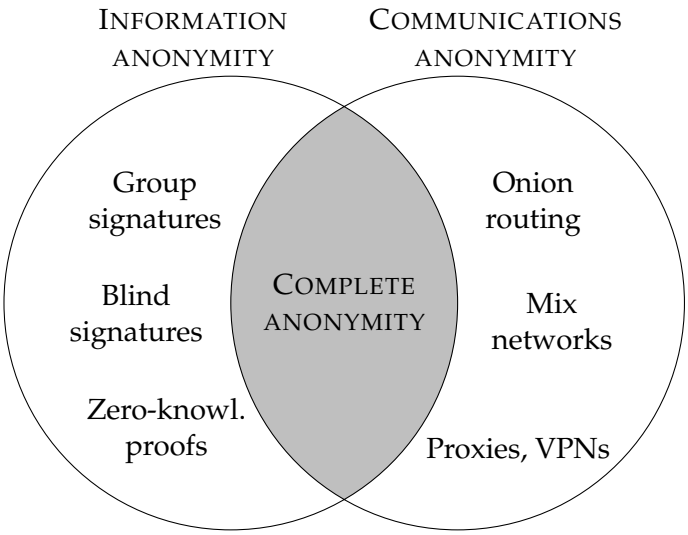


Figure 6.1: Cryptographic primitives like group and blind signatures ensure information anonymity; systems like mix and onion networks allow anonymous communications.

tionality for blocking traffic, including data coming from Tor and other anonymizing networks (BlockScript, offers a commercial service for blocking "unwanted" traffic and sells the raw blacklist data for $12,000 per year). Hence, it seems that in order for robust anonymity to achieve a wide acceptance, suitable solutions for addressing misbehavior must be found. Besides dishonest actions at the client side, server side illegitimate actions are being sheltered through anonymous networks. For instance, looking at what has received the name of *Deep web* [157], several illegal sites such as drug dealing markets and sexual abuse web pages may be found in a matter of minutes. An excerpt of drug related websites linked from the *Hidden Wiki*[3] is shown in Figure 6.2.

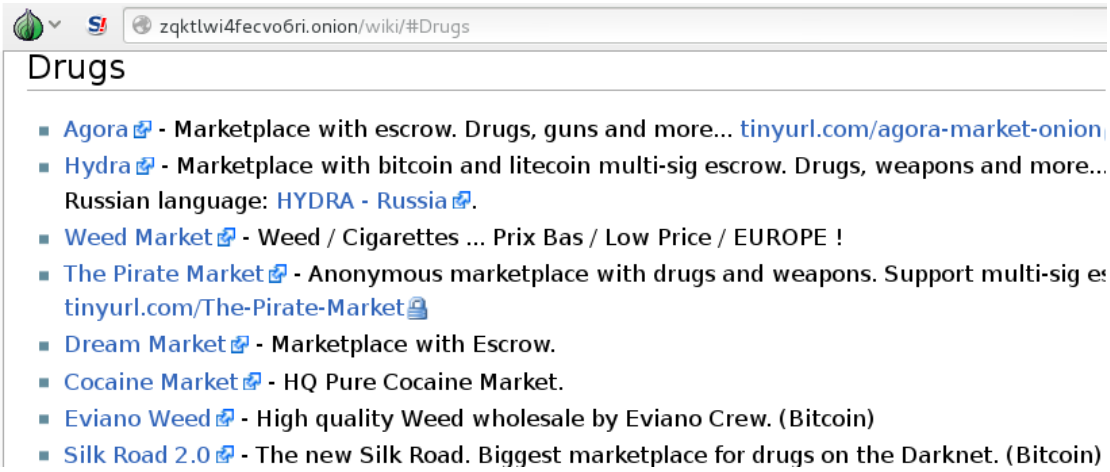


Figure 6.2: Excerpt of drug markets linked from the *Hidden Wiki*.

However, there are cryptography-based systems that allow reaching the intermedi-

[3] Available at `http://zqktlwi4fecvo6ri.onion/`. Accessed 11th May, 2014.

ate point. That is, systems where anonymity is still the main component, but that may be revoked in case of misbehavior, thus incorporating *fairness*. This kind of anonymity is referred to as *fair anonymity* (see Definition 10 for fairness).

At last, our second argument for the unsuccessful deployment of anonymity, is that when comparing fair anonymity cryptosystems with their equivalents dealing with, for instance, confidentiality or authenticity, there is a lack of supporting infrastructure. Taking as example the X.509 standard [185], a wide spread infrastructure exists that provides support for managing the trust relationship needed for incorporating authenticity requirements, which eases its adoption into the Internet. However, there is no equivalent and complete solution for the management of anonymity-based infrastructures.

In this chapter, we propose a complete solution to this task that allows a complete management of anonymous identities, based on the widely deployed X.509 infrastructure, and extending previous work by other authors [31]. Note that, once we are able to distribute anonymous identities by means of the protocol proposed in Chapter 5, we will be able to manage them with the extensions proposed in the current chapter. These extensions, in turn, will compose a building block for more complex anonymity systems, like the ones proposed in Chapter 7 and Chapter 8.

6.1 Related work

We may distinguish two areas of related work for this chapter. The first one encompasses the research performed in the subject of infrastructures for implementing fair anonymity. That is, any system which aims to be capable of being deployed over an existing anonymizing network or system in order to incorporate fairness into it. The second one comprises the standard mechanisms or proposals that may be used in order to deploy fairness in a standard friendly manner.

Fair anonymity systems. In the area of systems (with practicality in mind) for implementing *fairness* mechanisms, several approaches have been proposed based on blacklisting of misbehaving users. The first of this type of systems was Nymble [128, 181], which implements an efficient infrastructure that can be layered on top of Tor-like platforms, and enables blacklisting of users during predefined intervals of time. However, it places too much trust into two trusted authorities (the Pseudonym Manager and the Nymble Manager). Since then, many alternatives have been proposed in order to solve this problem. In BLAC [180], Service Providers (SP) keep their own blacklists against which users have to proof non-membership in order to gain access (hence eliminating the Nymble Manager authority). But in this case, given their construction, this increases the computational costs with respect to Nymble. In PEREA [24], the linear cost introduced in BLAC at the SP's side is eliminated, but it is shifted to the users, who must perform costly computations to demonstrate that they do not belong to the SP's blacklist. Jack [139], BNymble [140], Nymbler [114] and the extensions proposed in [112, 113] further refine these first approaches, by reducing the trust placed on the authorities while keeping the additional costs within manageable ranks. They also add new features, for instance Jack [139] allows users to generate their own authentication tokens, and allows the incorporation of objective blacklisting techniques (by means of *contracts*) proposed in [172]. The work in [112] further refines this later extension by making the revocation dependent on the specific contract that has been allegedly broken. However, the aforementioned blacklisting proposals are all highly customized systems, which complicates their incorporation into existing infrastructures and their adaptability to new specific needs that may arise even in the context for which they have been designed. In this section, we make a first proposal for addressing this *unadaptability* issue, by providing a means for revoking anonymity (at any degree possible), in an standard-like manner. Subsequently, in Chapter 8, we propose an extension to the Tor system for endowing it with a flexible and practical mechanism for fair anonymity.

Practical fairness management. The most extended PKI is undoubtedly *X.509* [185], which makes it also the basis for the majority of the identity-related management tasks over the Internet. X.509 specifies standard formats for public key certificates and a variety of extensions for managing them, abstracting out as much as possible the inner cryptographic details. Among the most important associated mechanisms, *Certificate Revocation Lists* or *CRL*s [185, Section 7.3]are data structures, in the shape of lists, containing serial numbers of X.509 certificates that have been revoked for some reason (compromise, expiration, etc.) and are thus no longer valid. This CRLs are static, in the sense that the CAs responsible for managing them periodically publish CRLs with updated information, but clients are not able to ask for specific certificates. For solving this problems, the *Online Certificate Status Protocol* or *OCSP* was created, [150] allowing

clients to ask about the status of any concrete X.509 certificate, at any moment in time.

Given the above, it would seem wise to adopt it, if possible, as a baseline for including functionality for anonymity management, in order to minimize costs and both ease and increase its deployability. Actually, the possibility for extending X.509 in such manner has already been done [31], using group signatures [65] as the underlying cryptographic primitive. The specification of X.509 certificates includes extensions that can be used to add functionality to the basic digital identity infrastructure. In [31], these extensions to X.509 certificates were used to incorporate anonymous digital identities. With the derived standard mechanisms, the ease of systems deployment increases notoriously. Certainly, *anonymous certificates*, i.e., certificates sustaining and endorsing the private/public keys of the underlying group signatures (introduced in Section 2.4.1) would be easy to incorporate into current infrastructures, and any required change would possibly be done with minimal effort.

It also is worth emphasizing that the X.509 standard is not the only option to handle anonymous digital identities. For instance, the ABC4Trust project [54] also promotes a standardized infrastructure to deal with Privacy Attribute Based Credentials (Privacy-ABCs). Like ABCs, X.509 also allows the use of attributes by means of Attribute Certificates (see [185, Section 3]). However, from the perspective of the *SDLC* it is convenient to use solutions granting usability, reliability, and scalability. Since general accepted standards have been thoroughly evaluated and subsequently validated with respect to those commitments, we propose to use the standard and already accepted X.509 infrastructure [185]. In other words, according to the principles of the SDLC our approach minimizes the impact on existing infrastructures by making use of currently deployed technology. It is worth noting that several proposals exist aiming to adapt group signatures into current standard infrastructures [52, 120–123], however they either do not use the X.509 infrastructure (which, as stated, is the most extended one) or do not explicitly deal with the issue of revocation which, as we will see, is essential for providing a complete functionality.

To conclude, it seems appropriate and necessary to develop a mechanism that allows incorporating fair anonymity into current systems. Moreover, such mechanism must not be dependable on any specific technology, like it is the case of the mechanisms explained in the first part of this section. As stated, X.509 is a natural candidate. However, even though quite useful X.509 extensions have been proposed in [31] for creating anonymous certificates, no extension is available for revoking them in case they are misused or compromised. Indeed, without suitable mechanisms for revoking anonymous certificates, no fairness may be provided. In the following sections we describe CRL and OCSP extensions for supporting such functionality. Additionally, due to the inherent properties of anonymity, and with the context set in [31], in order to revoke such an anonymous certificate, it does not suffice to just present the equivalent to a public certificate, and additional information must be provided. For addressing this issue, we also define in this chapter the *Anonymous Certificate Fairness Protocol* (*ACFP*).

6.2 CRL and OCSP extensions for anonymous certificates

In the same manner than the X.509 standard allows extension mechanisms for incorporating additional functionality to X.509 certificates, the same is possible for *Certificate Revocation Lists* (CRLs) and the *Online Certificate Status Protocol* (OCSP). By extending these mechanisms, which comprise the two main methods used for implementing revocation within X.509, we match the functionality available for X.509 anonymous certifi-

cates with that already being employed with conventional X.509 certificates. However, when referring to revocation within an anonymous setting, two types of revocation are possible: *unlinkability revocation* and *anonymity revocation* . The former type is applicable for anonymity schemes that prevent different actions performed by a same anonymous user to be linked among themselves. Thus, unlinkability revocation implies that those actions will no longer be unlinkable. On the other hand, anonymity revocation directly returns the real identity associated to the anonymous one. Therefore, it is a stronger revocation type with respect to privacy. If fairness is to be implemented through revocation, the two types must be accounted for. In this section we design extensions to CRL and OCSP for implementing this functionality. ASN.1 like definitions of these extensions are included in Appendix A.

6.2.1 Anonymity revocation with CRLs

A *Certificate Revocation List* is a list of X.509 certificates (more precisely, their identifiers) that have been revoked and are thus considered as no longer valid for the usage they were issued for. Whoever is interested in keeping an up to date list of these revoked certificates, needs to regularly request updates on these lists to the proper certification or revocation authority.

In order to extend the CRL mechanism for supporting revocation of unlinkability and anonymity, we must make use of the CRL entry extensions [185, Section 8.5.2]. Specifically, we need to add means for indicating the type of revocation that a CRL entry is dealing with. Hence, within the CRL entry extension depicted `revocationType` in Figure 6.3, we add a tag field for indicating the revocation type, which may take the following values and intended uses:

- `normal` : Conventional X.509 certificate revocation.
- `group` : Revocation of a complete group.
- `unlinkGroupMember` : Unlinkability revocation of group members.
- `anonGroupMember` : Anonymity revocation of group members.

The first two types will usually be equivalent in terms of the information they convey, but the `group` type emphasizes that it is dealing with a group certificate instead of a conventional one. The last two values will be used for publishing a sequence of revocation information of all the group members whose unlinkability or anonymity has been revoked.

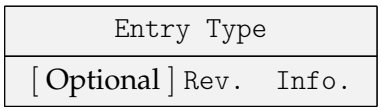


Figure 6.3: `revocationType` extension for CRLs. The optional `Rev. Info.` field will only be present when `Entry type` takes the values `unlinkGroupMembers` or `anonGroupMembers`.

Note that, depending on the value of the revocation type field, the extension will also include the information required for revoking the anonymity or unlinkability property of the corresponding group member. The specific contents of the `Rev. Info.`

field will vary according to the specific group signature scheme. For illustrative purposes, Figure 6.4 shows a sample of how an extended CRL would look like.

```
Certificate Revocation List (CRL):
    Version 2 (0x1)
  Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=US, ST=State, L=Locality1, O=Organization, OU=OrgUnit1,
            CN=Admin/emailAddress=admin@org.org
    Last Update: Jun 12 13:22:56 2012 GMT
    Next Update: Jul 12 13:22:56 2012 GMT
    CRL extensions:
        X509v3 CRL Number: 1
Revoked Certificates:
  Serial Number: 9B28ACA257136DE2
    Revocation Date: Jun 12 12:58:52 2012 GMT
    CRL entry extensions:
        entryType: unlinkGroupMember
        memberRevInfo: 7F:FF:FF:FF:FF:FE:30:E3
  Serial Number: 9B28ACA257136DE2
    Revocation Date: Jun 12 13:21:25 2012 GMT
    CRL entry extensions:
        entryType: unlinkGroupMember
        memberRevInfo: 7F:FF:FF:FF:FF:FE:29:F9
Signature Algorithm: sha1WithRSAEncryption
 85:7c:ef:46:fc:8f:d5:04:75:ba:49:56:a9:15:c0:ff:31:87:f1:ed:2e:17:84:3c:
 cc:2d:c1:51:c7:3c:da:d2:a8:cb:31:e2:16:95:f9:bf:92:0d:1c:03:43:f2:ef:2a:
 2a:fa:fb:65:8c:24:9f:8c:d0:11:cd:30:c3:36:f9:03:2a:0e:f3:54:45:10:a1:30
```

Figure 6.4: A sample of an extended CRL (the parts in bold show the differences with a classical CRL) based on the scheme in [132]. Note that the serial number of the two revoked members is the same, since they are both related to the same group.

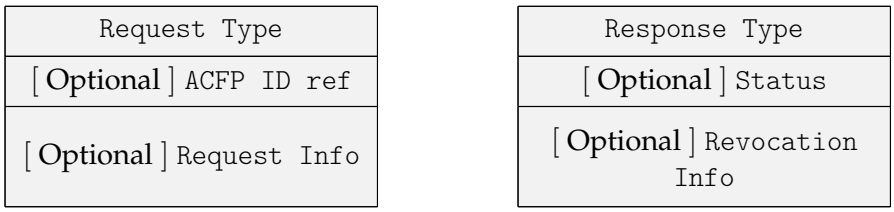
Finally, it is worth to emphasize that this extension would allow the implementation, following current standard technologies, of the revocation techniques named *Verifier-local revocation* in [45] and also suggested (and implemented) in [132].

6.2.2 Anonymity and unlinkability revocation with OCSP

The *Online Certificate Status Protocol* (OCSP) is a request-response protocol intended to achieve a more agile acquisition of information about the status of certificates than that provided by CRLs. Hence, it is necessary to extend both requests and responses to bear the revocation of anonymous certificates. OCSP includes support for adding this kind of extension using the `singleRequestExtensions` field [150, Section 4.1.1] for requests and the `singleExtensions` field [150, Section 4.2.1] for responses. Our proposal for the corresponding extension fields is sketched in Figure 6.5 and encompasses the `reqTypeInfo` extension for requests and the `rspTypeInfo` extension for responses.

The `Request Type` specifies what kind of information the requester wants to obtain. We include the following types for requesting different classes of status information:

- `normal` : for OCSP requests related to conventional X.509 certificates.
- `group` : to ask for the status of the group certificate (i.e. the group public key).



(a) `reqTypeInfo` extension for OCSP requests. The `ACFP ID ref` field will be present to reference evidence provided by means of ACFP. The `Request Info` field must be present for `Request Types` of `unlinkability`, `anonymity` or `groupSignature`.

(b) `rspTypeInfo` extension for OCSP responses.The `Status` field must be present for `Response Types` of `unlinkability`, `anonymity` or `groupSignature`. The `Revocation Info` will not be present for requests of type `normal`, `group` or `groupSignature`.

Figure 6.5: Extensions to OCSP requests and responses supporting anonymity and unlinkability revocation.

- `unlinkGroupMembers` : to demand the revocation information for all members with revoked unlinkability.
- `anonGroupMembers` : type for requesting the revocation information for all members with revoked anonymity.
- `unlinkability` : type for requesting the unlinkability revocation information of the issuer of a specific signature.
- `anonymity` : type for requesting the anonymity revocation information of the issuer of a specific signature.
- `groupSignature` : to request the status of the issuer of a specific (group) signature.

The first four types do not require any additional information other than a reference to the group being asked about (which is already included in current OCSP requests). The last three types, however, will require the inclusion of the optional field depicted as `Request Info` in Figure 6.5a. Normally it would suffice to include a group signature, although it might also be necessary to add some extra information. Hence, its internals will depend on the specific group signature scheme and context. The `ACFP ID ref` field will be used most frequently with requests of the three last types, when evidence exists that may have recently caused a change in the status of the group signature included within the `Request Info`. Its usage will become clearer when we explain ACFP in Section 6.3. Figure 6.6 depicts a sample of an extended OCSP request.

For responses, depicted in Figure 6.5b, the corresponding response types need to be added. Additionally, the `Status` field will be present when the response is of any of the three last types in the previous list. Finally, the `Revocation Info` field may take different shapes depending on the response type. It will not be present for `normal`, `group` or `groupSignature` responses (they only inform about the status). For the `unlinkGroupMembers` and `anonGroupMembers` types, it will be a sequence containing the revocation information of all the members of the group whose unlinkability or anonymity has been

```
OCSP Request Data:
 Version: 1 (0x0)
 Requestor List:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 49FB20671058982612F8CE969CC9070B3012DF6D
      Issuer Key Hash: 87213371C00256ABEFF3CA6BB16078ED5499FC3B
      Serial Number: 9B28ACA257136FB7
 Request Extensions:
     OCSP Nonce: 0410221BC72A6359A7DBB8CCEB49AAF1F747
 Single Response Extensions:
  reqTypeInfo:
   reqType: groupSignature
   reqInfo:
    33:82:0b:59:e2:1e:49:ef:e1:80:e6:e4:c7:7e:f5:cf:20:6a:72:b3:bd:8e:47:02:b5:9b:01:71:e8:
    36:1b:0b:60:35:f0:d6:a0:cc:96:de:e6:77:e0:96:89:fc:47:6a:07:97:89:40:5d:02:9e:2b:31:52:
    34:19:01:ae:94:f6:46:05:b7:d2:4b:f8:31:70:d1:5a:c9:00:65:94:e1:63:f6:03:98:8f:98:8e:7c:
    46:f7:19:6f:8a:3a:9e:54:a2:d1:66:4f:f2:80:60:48:c5:26:df:ed:a6:a3:b1:50:25:64:05:64:b0:
    e9:0c:f0:26:0e:fb:4f:31:97:b1:1c:dd:4e:f8:cb:84:69:e4:94:5a:d8:26:50:12:2a:c9:21:18:2c:
    ca:57:be:28:b5:b8:98:62:b1:73:b3:45:7d:11:f1:e0:8a:a8:ba:7a:ae:9a:94:3f:ce:77:c6:76:ab:
    fd:de:6b:a9:e5:5b:55:c9:2e:a2:72:b0:60:13:36:2d:aa:a2:24:3f:b3:04:72:06
```

Figure 6.6: A sample of an extended OCSP request of `reqType` *groupSignature* (the parts in bold show the differences with a classical OCSP request).

revoked, respectively. For `unlinkability` and `anonymity` types, and in case `Status` indicates that the associated member has been revoked, it will contain one single entry (instead of a sequence) with the corresponding revocation information. Again, the contents of each specific revocation information token(s) will vary depending on the related group signature scheme. Figure 6.7 and Figure 6.8 show samples of extended OCSP responses for types `groupSignature` and `groupMembers`, respectively.

```
Response verify OK
0x9B28ACA257136FB7: good
    Request Extensions:
        OCSP Nonce: DE64EFA69AC9E0B3984CADBF2B4E2F94E6D1
    Single Extensions:
        rspTypeInfo:
            rspType: groupSignature
            rspStatus: revoked
        This Update: Jun 12 13:23:25 2012 GMT
        Next Update: Jun 12 20:13:00 2012 GMT
```

Figure 6.7: A sample of an extended OCSP response of `rspType` *groupSignature* (the parts in bold show the differences with a classical OCSP request). Note that the status of the group certificate reflects that the group is still valid, but the *rspStatus* field shows that the private key used to issue the given signature has been revoked.

At last, note that it is possible to define new types of requests and responses as the need arises, following the same structure, with the necessary new types definitions.

```
Response verify OK
0x9B28ACA257136DE2: good
    Request Extensions:
        OCSP Nonce: 78AE9E2F01E7C3E8A1031CADB7fB80E725B9
    Single Extensions:
        rspTypeInfo:
            rspType: unlinkGroupMembers
            Revoked members Information:
              memberRevInfo: 7F:FF:FF:FF:FF:FE:30:E3
              memberRevInfo: 7F:FF:FF:FF:FF:FE:29:F9
        This Update: Jul 30 14:32:10 2012 GMT
        Next Update: Jul 30 20:15:00 2012 GMT
```

Figure 6.8: A sample of an extended OCSP response of `rspType` *groupMembers* (the parts in bold show the differences with a classical OCSP request). Note that the status of the group certificate reflects that the group is still valid, but one member revocation information field is added for each revoked member key associated to that group.

For instance, it could be useful to include *delta OCSPs*[4] for each of the `GroupMembers` requests types.

6.3 ACFP: Anonymous Certificate Fairness Protocol

With the extensions to CRL and OCSP proposed in Section 6.2, we allow the revocation of both anonymity and unlinkability using standard friendly mechanisms. However, given the sensitivity of any kind of revocation in scenarios where anonymity is of importance, it may be required to send additional evidence to an authority in order to revoke either unlinkability or anonymity. To the best of our knowledge, no current protocol exists supporting that functionality, which hinders the deployability of fair anonymous systems [76]. To that end, we define here the *Anonymous Certificate Fairness Protocol* (*ACFP*). Given the wide variety of contexts in which it could be applied, we aim to define it as flexible as possible. This means that besides defining the necessary information for guaranteeing its correctness, security and any other required meta information, we do not define the contents of any specific query, and just leave it as an abstract field within a complete message. For that matter, different kinds of evidence shall be required depending on the scenario and the specific policies to perform a revocation.

The proposal of a completely new protocol could be read as a contradiction with respect to our objective of reusing current technology as much as possible, according to our *pragmatism principle* stated in Chapter 1. However, the well known encapsulation and modularity principles for the design of protocols and programming interfaces demand to identify functionalities and implement them independently [131]. In our case, we could include the functionalities in ACFP to the current OCSP. Nevertheless, this option makes OCSP a more complex protocol that addresses two different problems. In other words, we incur a violation of the previously commented principle for the design of protocols and programming interfaces. In addition, as we will show next,

[4]That is, *partial* OCSP responses that contain only the information related to the group members that have been revoked since the last query or a specified instant.

we have followed the design principles of OCSP, what makes our proposal completely suitable to be used within the same technological scenario based on X.509.

ACFP should allow clients to send any kind of evidence to a suitable authority. Typically, a client (e.g. some service provider) that thinks that some misbehavior has occurred related to an anonymous identity, will ask about the status of that identity (through any means within those defined in Section 6.2). If the identity is not within the revoked ones, then it will first send the required evidence to prove the misbehavior, and then an OCSP request will be made by referencing the related ACFP conversation. Note however that the client providing evidences does not need to be the same who requests the status of a certificate. Alternatively, the ACFP authority could periodically send *revocation orders* to CRL/OCSP authorities, related to group members for which enough evidence has been provided recently.

Note that we are assuming that there is some kind of policy that enables authorities to make a decision given a set of evidences. Thus, the client must also include a reference to a suitable policy, in order for the authority to be able to evaluate whether or not a misbehavior has occurred according to that policy. In turn, this implies that different policies may require different types of evidences. Figure 6.9 shows our proposal for transmitting specific evidences. The `Evidence ID` would be the same for all evidences that are related to the same misbehavior, allowing to reduce costs and protocol complexity by creating a "case" and reference it afterwards either from ACFP or from OCSP requests (see Section 6.2.2). When no previous evidence has been sent, the ID field can be left to `NULL`, and a new ID will be assigned in the associated response. The `Request type` field and `Policy ID` fields establish a context for the current evidence: the former indicates the aim of the current request, while the second specifies which policy is related to it. For the `Request type` tag field, we define three possible values:

- `evidence` : type used for requests intended only to provide evidence of a misbehavior.
- `unlinkability` : used to ask whether or not there is enough evidence (for the case specified in `Evidence ID`) to execute an unlinkability OCSP request.
- `anonymity` : to ask whether or not there is enough evidence (for the case specified in `Evidence ID`) to execute an anonymity OCSP request.

The optional `Evidence data` field contains the evidence data itself, whose syntax and semantics would vary depending on the specific group signature scheme and application context. Additionally, it will only be present when the request is of type `evidence`. Finally, the `Single Evidence Extensions` field is an optional field intended to add any new functionality that may be required.

For instance, a real evidence structure for an unlinkability request might be as shown in Figure 6.10. Upon receiving that evidence, and if deemed satisfactory, the suitable authority would use the group signatures received within the evidence in order to revoke the unlinkability property of the misbehaving signer.

One or more evidence structures may be included within a single ACFP request, as shown in Figure 6.11. Besides them, a complete ACFP request is composed by: a field indicating the protocol version; a field with the identity of the requester; optional extensions; and a signature of the whole request. The signature for the requests is made optional to enable any party to provide evidence of misbehavior in scenarios that support it. However, it is up to the server to make it compulsory.

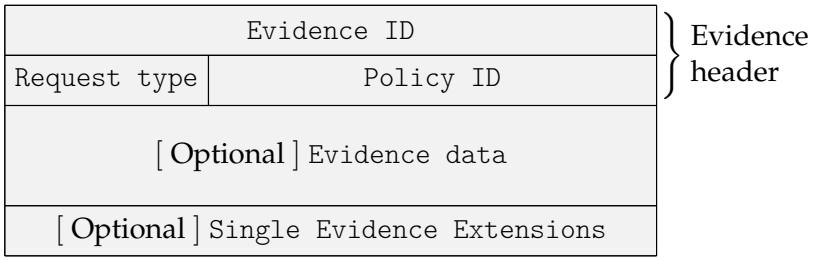


Figure 6.9: Single evidence structure. The `Evidence data` field is present for requests of type `evidence`. Extensions to the protocol may be included using the optional field `Single Evidence Extensions`.

```
Evidence ID: 654887
Request type: Unlinkability
Policy ID: 57
Evidence data:
  Group signature:
    33:82:0b:59:e2:1e:49:ef:e1:80:e6:e4:c7:7e:f5:cf:20:6a:72:b3:bd:8e:47:02:b5:9b:01:71:e8:
    36:1b:0b:60:35:f0:d6:a0:cc:96:de:e6:77:e0:96:89:fc:47:6a:07:97:89:40:5d:02:9e:2b:31:52
  Group signature:
    34:19:01:ae:94:f6:46:05:b7:d2:4b:f8:31:70:d1:5a:c9:00:65:94:e1:63:f6:03:98:8f:98:8e:7c:
    46:f7:19:6f:8a:3a:9e:54:a2:d1:66:4f:f2:80:60:48:c5:26:df:ed:a6:a3:b1:50:25:64:05:64:b0
  Group signature:
    e9:0c:f0:26:0e:fb:4f:31:97:b1:1c:dd:4e:f8:cb:84:69:e4:94:5a:d8:26:50:12:2a:c9:21:18:2c:
    ca:57:be:28:b5:b8:98:62:b1:73:b3:45:7d:11:f1:e0:8a:a8:ba:7a:ae:9a:94:3f:ce:77:c6:76:ab
```

Figure 6.10: Sample evidence structure.

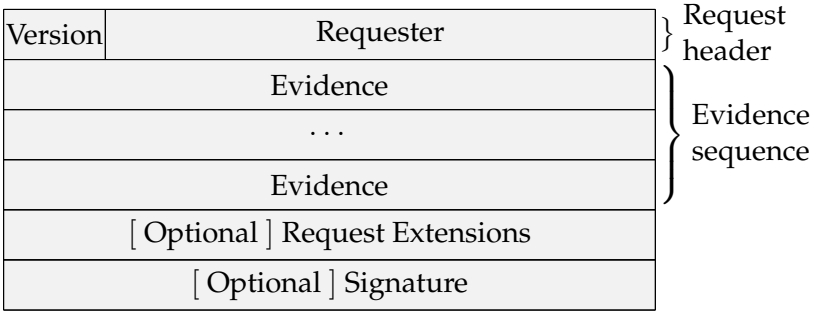


Figure 6.11: ACFP request composed by several evidences. Extensions to the protocol may be included using the optional field `Request Extensions`. Servers may demand signed requests, case in which the field `Signature` must contain a valid signature of the whole request.

Responses to each individual request will follow the structure depicted in Figure 6.12. The `Evidence ID` field serves the same purpose than in the request (in case the ID was set to `NULL` in the request, a new ID will be generated and included here). The evidence response status field indicates the output obtained after processing the request, and it will depend on the specific type: the different combinations are shown in Table 6.1. Basically, besides the conventional error statuses, an `evidence` response status means that there is not enough evidence to either grant or deny any possible subsequent OCSP revocation request (of anonymity or unlinkability type). On the other

hand, the `accepted` and `denied` statuses accept or deny what is being requested, respectively.

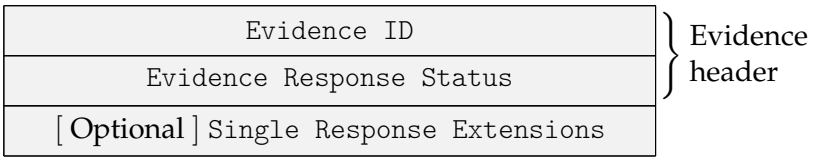


Figure 6.12: Single evidence response structure. Extensions to the protocol may be added by making use of the optional field `Single Response Extensions`.

A real ACFP single evidence response might look as shown in Figure 6.13.

```
Single Response:
  Evidence Set ID: 654887
  Evidence Response Status: accepted
```

Figure 6.13: Sample single evidence response structure.

Req. type	evidence	anonymity	unlinkability
Response Status	Malformation		
	UnknownID		
	UnknownPolicy		
	OK		
		evidence	
		accepted	
		denied	

Table 6.1: Possible evidence response status depending on the request type. A shaded cell indicates that the response status associated to that row is not possible for the request type in the corresponding column.

In the same manner than ACFP requests, several single responses will be grouped together, keeping the same relative ordering with respect to the associated request, in order to conform a sequence of responses. A `version` field indicating the protocol version, and a global response status (do not confuse with each single response status) compose the response header. Also, optional response extensions and signature field may be present. The complete structure is shown in Figure 6.14.

Once defined the functionality provided by ACFP, and knowing its relationship with OCSP, we show in Figure 6.15 and Figure 6.16 the flow diagrams that server and client (respectively) shall follow to solve typical requests. Note that, for simplicity, this diagrams assume that the role of OCSP and ACFP authority is played by the same entity. In a realistic scenario, this roles should be separated, and the communications between them, secured.

For instance, an OCSP server receiving an request with the `reqTypeInfo` extension with `reqType` set to `groupSignature` (see Section 6.2.2), where the associated key is not already present in a CRL, and with a reference to an ACFP conversation with enough evidence supporting the request, would act as follows:

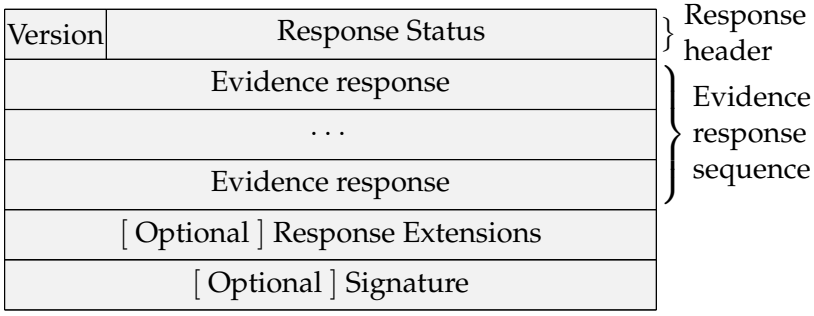


Figure 6.14: ACFP response composed by several individual responses. Extensions to the protocol may be included using the optional field `Response Extensions`. Servers may send signed requests, case in which the field `Signature` will contain a valid signature of the whole response.

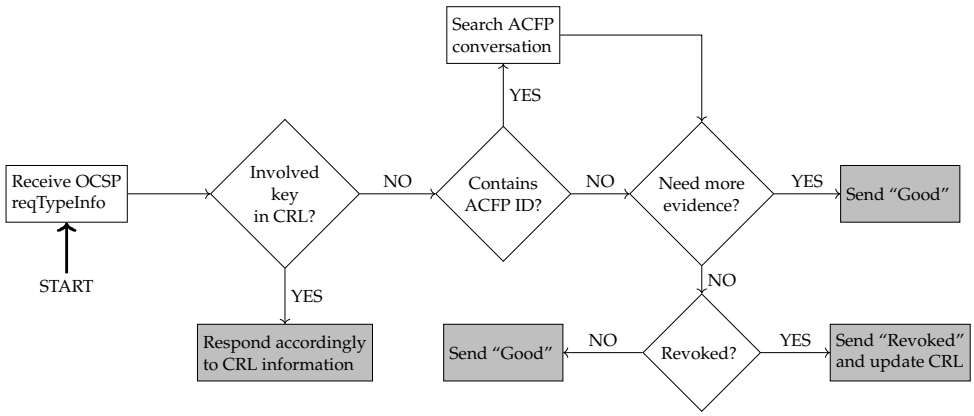


Figure 6.15: Server flow diagram defining the relation between OCSP and ACFP for evidence supported revocation requests.

1. Look up in the CRL (using the tracing functionality, see Section 2.4.1: fail.
2. Get the `ACFP ID ref` field of the request and search the associated conversation.
3. Check whether the referenced ACFP evidence is enough for granting the request.
4. Since there is enough evidence, update the CRL with the signer's tracing trapdoor and answer the requestor with `revoked`.

From the client perspective, the relation between ACFP and OCSP may be captured as follows:

1. A client may directly perform an OCSP query, without referencing any ACFP evidence. In that case, the server will see if the involved certificate has been somehow revoked, or if there is any ACFP evidence related to it.
2. A client may first provide ACFP evidence concerning some misbehavior. When it receives the ACFP identifier, the client may perform an OCSP query referencing that identifier. The server will perform the same actions as before, but also considering the referenced evidence.

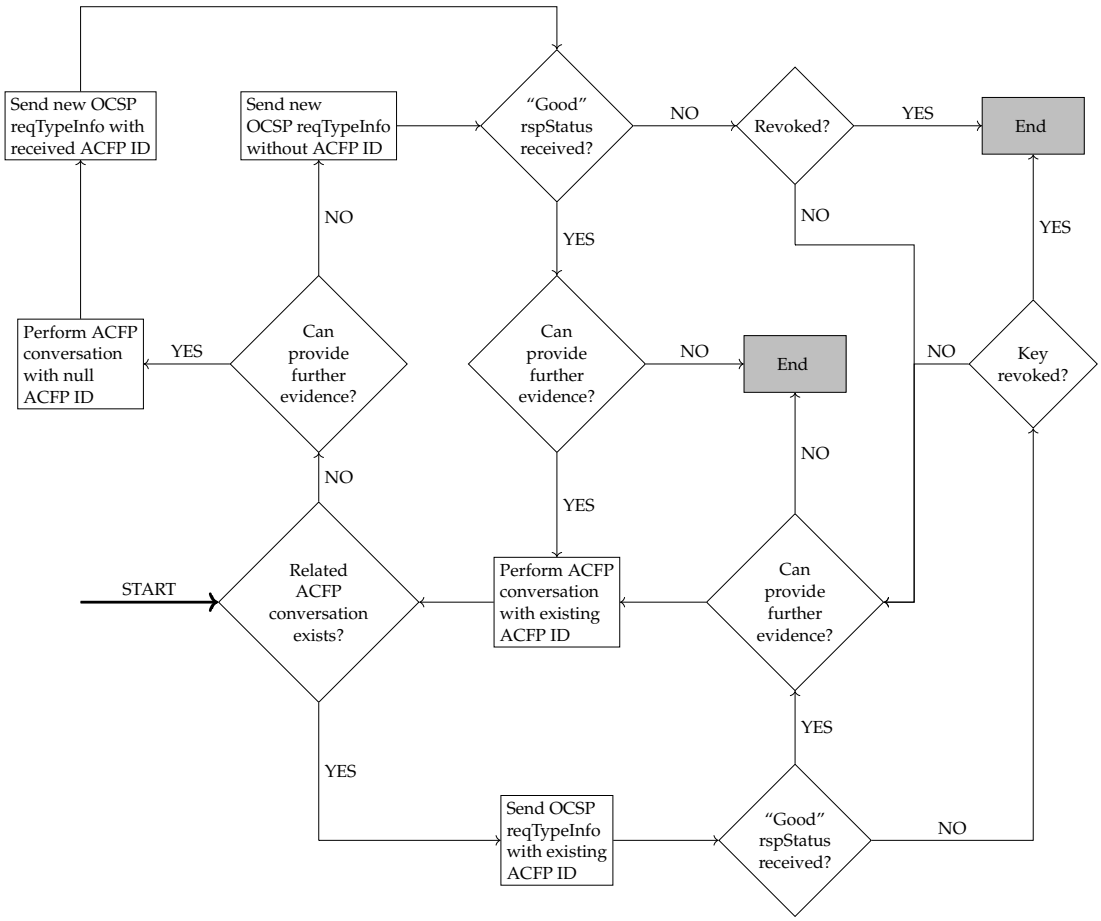


Figure 6.16: Client flow diagram defining the relation between OCSP and ACFP for evidence supported revocation requests.

3. A client may just provide ACFP evidence to denounce some misbehavior, so that the revocation authorities take the appropriate measures.

Finally, ASN.1 like definitions of request and responses in ACFP are given in Appendix A.

6.3.1 Policies

We have mentioned policies that take part into the decision on granting or denying a specific revocation request. In light of [76], it is clear that policies take a central role when it comes to provide *fairness* by the technological implementation of the legal requirements that may arise in the specific application contexts. Naturally, some policies are not subject to be processed in an automated way, and would have to be processed manually[5]. However, many policies can be represented as a set of rules, procedures, or some other method of automated reasoning. Authorities providing *fairness* can publish the policies upon which they will accept requests for any of the revocation types they support. If, according to some policy, enough evidence has been provided via ACFP,

[5]Note that, even in this case, ACFP is still required, since it is the way to provide (electronic) evidence in an anonymized scenario.

when an OCSP request arrives referencing that ACFP conversation, the revocation shall be performed. Requests for which there is not enough evidence shall be discarded.

Example. Anonymous certificates and their revocation may be useful in any Internet based communication system. For instance, a forum in a web platform. The forum administrator could require its users to anonymously sign all the messages they post with a fair traceable group signature scheme (e.g., [32]). In case it occurs some kind of misbehavior, the administrator could require a suitable authority to revoke the anonymity or unlinkability of the author of a set of messages, if they were indeed sent by the same signer. Using our protocols, this action would require the administrator to send the evidences to a suitable ACFP authority. If there is enough evidence, the ACFP authority asks the OCSP authority (probably referencing the previous ACFP conversation) to revoke the misbehaving party.

The configuration of the revocation scheme has as a first step the publication of a suitable policy by the ACFP authority. A simple policy is shown in Figure 6.17. It states that, if there is evidence proving that the same signer has sent more than a predefined threshold of messages within a predetermined time span, the unlinkability (or anonymity) property of that member will be revoked.

```
#  Input: array of size n of messages
#         ordered by date.
#  Output: accepted, evidence or denied.
if reqType = anon && n < ANON_N_LIMIT
  return ''evidence''
if reqType = unlink && n < UNLINK_N_LIMIT
  return ''evidence''
if have_same_signer
  if reqType = anon
    if timespan <= ANON_TIMESPAN_LIMIT
      return ''accepted''
  if reqType = unlink
    if timespan <= UNLINK_TIMESPAN_LIMIT
      return ''accepted''
return ''denied''
```

Figure 6.17: Sample policy for revocation due to spamming.

With this scenario in mind, take as example the case of combining the configuration where only subsidiary authorities can trace misbehaving users (and other users only learn the status of a certificate) and a group signature scheme offering the same functionality as [32] is adopted. Suppose now that a service provider sends a set of `UNLINK_N_LIMIT` messages, sent via the same SSL session, to an ACFP server. Since all the messages come from the same SSL session, which was negotiated using an anonymous identity at the client's side[6], all the subsidiary authorities may conclude that there is enough evidence to revoke the unlinkability of the originator. They consequently send their shares to the master server, who will proceed to revoke the user's unlinkability property and contact the appropriate OCSP server for updating its CRL. When the revoked user tries to establish a new connection with some service provider, the latter would proceed to verify the associated private key, via the received group

[6]This would indeed be possible, using anonymous X.509 certificates!

signature. Having the unlinkability of the former being revoked, the response would be `unlinkRevoked` (or just `revoked`). Note that this does not even degrade the revoked user's anonymity to pseudonymity, since it is just associated to the `revoked` status.

6.4 Security considerations

An important step in the development of a security protocol is the identification of the critical components along with the security properties required to each of these elements. In our case, the information conveyed using the proposed extensions to OCSP and with ACFP is highly sensitive. Thus, *confidentiality*. Moreover, confidentiality should be assured by impeding meaningful information leakage from the analysis of the ciphertexts. Certainly, ciphertext indistinguishability is required since otherwise an attacker could identify revoked users by comparing the transmitted revocation information (despite it being encrypted). Even though revoked users have seen their privacy rights reduced due to policy execution, it may not be desired for outsiders to learn which users are revoked. Additionally, the attacker would successfully differentiate revoked users from non-revoked ones due to the different sizes in the responses (issue solved with ciphertext indistinguishability).

Besides confidentiality, authenticity is at least desirable for responses. This is necessary to avoid *spoofing* attacks in which an attacker introduces a fraudulent response to specific requests (e.g. changing `good` for `revoked` in `groupSignature` requests and thus, performing an effective Denial-of-Service on the victim, or by changing any of the referenced identifiers in order to mislead the requester). For requests, the authorities may support receiving un-authenticated messages. This may be desirable when they want anyone to provide evidence of a misbehavior. Note that this does not affect the correctness of the system: if some provided information is not valid, the authority just rejects it, being then the properties of the underlying group signature scheme used to represent the anonymous identities responsible for ensuring the mentioned properties. Finally, for authenticity, ACFP supports optional signatures to be included in both requests and responses. As for OCSP, it is possible for authorities to demand signed requests too.

Besides these considerations, note that both OCSP and ACFP are implemented as single exchange protocols (i.e., the protocol flow is composed just by a request and a response). Thus, other than requiring confidentiality, authenticity and integrity, no further requirement or formal proof seems necessary. For this purpose, we assume that both OCSP and ACFP are layered on top of SSL/TLS with an adequate configuration. As for additional security properties (e.g. robustness against misidentification or framing attacks [132] or distributed trust [32]), these would be directly inherited from the chosen group signature schemes, as stated above, and our protocol does not impose additional limitations on their adoption.

6.5 Experimental evaluation

In this section, we show some experimental results obtained with a simple setting. We start by describing several possibilities for configuring a *fairness infrastructure* using the proposed extensions. Next, we introduce the specific details of our testing environment and the chosen cryptographic building blocks, which also have a direct impact in the final computational and implementation costs. Then, we detail what information we

have taken into account and how it is composed. Finally, we comment the obtained results.

Based solely on the possible configurations of our proposal (i.e., ignoring the underlying group signature primitive) several alternatives are possible. For instance, one option would be for the ACFP server to be used just to store and process evidences: the OCSP server would then ask about these specific evidences, when referenced by common users, and use them to update the related certificates. On the other hand, the ACFP authority could also hold the private information necessary to revoke members. In that case, it would be the ACFP who actually revoked the certificates, and then transmitted to the OCSP server only the status and all the tracing or identifying information. Also, it is possible to offer different degrees of privacy by limiting the supported types of request/responses depending on the originator of communications. For instance, allowing anyone to obtain the revocation information of the members of a group might be considered privacy threatening (even if there is enough evidence justifying revocation). Hence, the revocation authorities could decide to give that kind of information only to trusted subsidiary authorities, while only responding requests of type `groupSignature` (see Section 6.2.2) to the common public.

Even more flexibility (from the functionality perspective, and consequently from the privacy perspective too) is provided by the fact that our proposal does not pose any requirement on the underlying group signature technique, hence it directly supports any privacy protecting technique compatible with group signatures. For instance, a trust distribution strategy where authorities only own shares of the revocation keys could be employed by making use of group signature schemes like the one proposed in [32]. In that case, each authority holding a share of the revocation key must participate in the revocation process in order for it to be completed. Using the ACFP protocol, this "subsidiary authorities" would just have to send an ACFP request to the corresponding "master ACFP server", with the associated evidence and their corresponding share. Other alternatives exist besides this trust distribution mechanisms. For example, group signatures allowing unlinkability revocation, but not anonymity revocation, could be applied if revealing the members' identities is not admissible in some specific context. Thus, systems can be configured on demand according to concrete legal/technological needs in order to balance privacy and accountability.

Testing configuration. In short, we have implemented a simple OCSP server which just responds to our extended requests, and a simple ACFP server that only knows the policy shown in Figure 6.17. And of course, their corresponding clients. For the tests, we used a group with 5000 members. 20% of these members were revoked from the beginning (that is, not due to a policy execution)[7]. As specific values for the implemented policy (Figure 6.17) we set `ANON_N_LIMIT` and `UNLINK_N_LIMIT` to 10 and 5, respectively; and `ANON_TIMESPAN_LIMIT` and `UNLINK_TIMESPAN_LIMIT` both to 60 seconds. In other words, 10 messages sent in less than 60 seconds are enough for granting an anonymity revocation request, and 5 messages in less than 60 seconds are enough for granting an unlinkability revocation request. As for the specific contents of the requests and responses, for the OCSP extensions, `reqInfo` consists on messages and their corresponding group signatures, and `memberRevInfo` is composed by an integer for granted revocations. For ACFP, `Evidence data` is also a set of messages with their group signatures, and the response is the appropriate code. All the tests have been per-

[7] This is important to be realistic, since in [132] the cost of signature verification is linear in the number of revoked members, if we want to check if they have been issued by a non-revoked member.

formed in a Quad Core Intel i7, with 3.40 GHz and 16 GB of RAM running Debian 7.5 with Linux kernel 3.2. Both servers (ACFP and extended OCSP) and the clients were located in the same machine (hence, the communication time is negligible) and have been implemented in C.

Cryptographic building blocks. In order to illustrate the flexibility of our proposal, we have implemented a prototype supporting two different group signature schemes. The first group signature scheme is the one defined in [132] and noted as KTY04, whereas the second one is given in [67] and hereafter referred as CPY06. Both of them have been incorporated into the prototype through the *libgroupsig* library described in Chapter 4, and they both support tracing, required for handling privacy respectful unlinkability requests. Note however that there are other schemes that may improve some aspects of them, like the computational costs of the different operations (see the systems described in [137, 138] as possible alternatives). An introduction to group signatures is available in Section 2.4.1. Additionally, further experiments on different group signatures are included in Chapter 4. As for their configuration, we have used a group modulus of 1024 bits for KTY04 and a group order of 160 bits for CPY06 (according to the NIST[8] Elliptic Curve Cryptography (ECC) with 160-bit numbers provide roughly the same security level than "non ECC" with 1024-bit numbers).

Experiments description. Several types of queries have been made, for testing both the OCSP extensions and ACFP. The obtained results are shown in Table 6.2 and Table 6.3 where, within each cell, we include the following information:

- First line: average number of seconds required to attend a request.
- Second line: average number of bytes per request.
- Third line: average number of bytes per request that were used to transmit any of the anonymous signatures included within the requests (including the request signature itself). This is an important value to consider how much improvement would be possible on the overall costs when using more efficient anonymous signature schemes.
- Fourth line: average number of bytes per response. These responses include the corresponding trapdoors in case of granted revocations.

In the case of OCSP, we are interested in measuring the additional costs introduced by the extensions. For this purpose, we have performed the following queries, obtaining the results shown in Table 6.2.

1. **reqType** equal to `groupSignature` for signatures issued by unrevoked members. Therefore, these requests originated a response with **rspStatus** of `good`. During these requests, a CRL look-up is performed.
2. **reqType** equal to `groupSignature` for signatures issued by previously revoked members. These requests originated a response with **rspStatus** of `revoked`[9]. During these requests, a CRL look-up is performed.

[8] `http://www.nsa.gov/business/programs/elliptic_curve.shtml`. Last access on March 31st, 2015.
[9] Our OCSP server does not differentiate anonymity and unlinkability revocations.

3. **reqType** equal to `unlinkability` (resp. `anonymity`) for signatures issued by unrevoked members, and providing an *incomplete* ACFP conversation reference (i.e. a conversation with not enough evidence). Hence, this requests were responded with `good`. Within this OCSP requests, ACFP requests from the OCSP server to the ACFP server were performed to check the validity of the referenced ACFP evidence. For unlinkability requests, a CRL look-up is performed before consulting the ACFP server.

4. **reqType** equal to `unlinkability` (resp. `anonymity`) for signatures issued by unrevoked members, but providing a *complete* ACFP conversation reference (i.e. a conversation with enough and valid evidences). This requests caused the signer of the signature sent within the OCSP request to be added to the CRL (hence revoked), and where answered with a `revoked` **rspStatus**, along with the tracing trapdoor (for unlinkability revocations) or the signer's ID (for anonymity revocations). Within this OCSP requests, ACFP requests from the OCSP server to the ACFP server were performed to check the validity of the referenced ACFP evidence. For unlinkability requests, a CRL look-up is performed before consulting the ACFP server.

From the tables, two facts are worth to be emphasized. First, the communication costs in terms of transmitted bytes decreases when using CPY06 to roughly 1/3 (and sometimes even more). But secondly, the computational costs are worse, sometimes even 25 times higher. These results are inherent to the employed schemes. CPY06 uses ECC, thus producing much smaller group signatures. However, while in KTY04 most of the operations are just performed using "simple" modular exponentiations, in CPY06 some of them imply the much more time-consuming bilinear pairings. This also shows the flexibility of different group signature schemes and, as mentioned above, more crafted implementations of all the involved components will certainly produce much better results. To establish a baseline with respect to conventional OCSP requests, recent reports state[10] that the latency of OCSP queries may be approximately in the interval of 0.03 and 0.6 seconds. Looking at the results obtained for our extension to OCSP, and keeping in mind that our experiments have been performed through a prototype (thus, barely optimized), the comparison is promising. Depending on the type of extended request, we have obtained an average latency time between 0.015 and 3.80 seconds (note however that our experiments did not take into account the roundtrip time, as they were all performed locally). 0.015 seconds are basically in line with conventional OCSP requests; as for the maximum latency of 3.80 seconds (obtained for `unlinkability` requests with CPY06), it will probably be greatly reduced through an optimization of the implementation of the underlying mathematical operations, and through an optimization of the users' groups and server equipment.

In order to evaluate ACFP by itself, we have performed the ACFP requests shown below considering that the ACFP server first verified if the received evidence was correctly composed (the group signatures valid) and then applied the sample policy (Figure 6.17):

1. **reqType** equal to `unlinkability` (resp. `anonymity`), without providing any evidence at all. Hence this requests were responded with an `evidence` **rspStatus**.

[10] See, for instance, the analysis by Symantec or Netcraft, at `http://www.symantec.com/connect/blogs/what-ocsp-0` (last access, March 23rd, 2015) and `http://news.netcraft.com/archives/2013/04/16/certificate-revocation-and-the-performance-of-ocsp.html`, (last access, March 23rd, 2015) respectively.

OCSP			
KTY04		**rspStatus**	
		`good`	`revoked`
reqType	`groupSignature`	0.160 secs/request ≈ 2734 bytes/request ≈ 2723 signature bytes 15 bytes/response	0.084 secs/request 94 bytes/response
	`unlinkability`	0.172 secs/request ≈ 2742 bytes/request ≈ 2723 signature bytes 20 bytes/response	0.241 secs/request 92 bytes/response
	`anonymity`	0.015 secs/request ≈ 2738 bytes/request ≈ 2723 signature bytes 16 bytes/response	0.019 secs/request 21 bytes/response
CPY06		**rspStatus**	
		`good`	`revoked`
reqType	`groupSignature`	≈ 2.23 secs/request ≈ 864 bytes/request ≈ 856 signature bytes 15 bytes/response	 18 bytes/response
	`unlinkability`	2.55 secs/request ≈ 872 bytes/request ≈ 856 signature bytes 20 bytes/response	3.80 secs/request 120 bytes/response
	`anonymity`	0.03 secs/request ≈ 868 bytes/request ≈ 856 signature bytes 16 bytes/response	0.04 secs/request 21 bytes/response

Table 6.2: Experimental results for the extended OCSP tests using KTY04 (top) and CPY06 (bottom). For requests of type `groupSignature`, we queried about unrevoked keys for the data shown under the column `good`, and previously revoked keys for the data shown under the column `revoked`. In both cases, the field **ACFPRef** was unused. For requests of type `unlinkability` and `anonymity`, we queried about unrevoked certificates, referencing enough evidence for the tests shown in column `revoked` (hence, the corresponding keys were revoked) and with not enough evidence for the tests in column `good` (leaving the corresponding keys valid). All values are averaged over 1000 requests/responses.

2. **reqType** equal to `unlinkability` (resp. `anonymity`), providing `UNLINK_N_LIMIT` (resp. `ANON_N_LIMIT`) evidences, but with timestamps of more than `UNLINK_TIMESPAN_LIMIT` (resp. `ANON_TIMESPAN_LIMIT`) seconds away. Thus, this requests were responded with a `denied` **rspStatus**.

3. **reqType** equal to `unlinkability` (resp. `anonymity`), providing `UNLINK_N_LIMIT` (resp. `ANON_N_LIMIT`) evidences, with timestamps of less than `UNLINK_TIMESPAN_LIMIT` (resp. `ANON_TIMESPAN_LIMIT`) seconds away. Thus, this requests were responded with an `accepted` **rspStatus**.

The obtained results are shown in Table 6.3. According to our protocol, all the ACFP requests were signed anonymously (using either [132] or [67]). Note that the `denied`

and `accepted` columns mostly include the same information (except the bytes/response row). Specifically, the same number of evidences (composed by messages and their group signatures) are sent by the requester and processed by the responder. However, only in those in the `accepted` meet the conditions in Figure 6.17 for granting a revocation.

ACFP				
KTY04		**rspStatus**		
		`evidence`	`denied`	`accepted`
reqType	`unlink.`	0.015 secs/request 2757 bytes/request 2701 sign. bytes 26 bytes/response	≈ 0.057 secs/request ≈ 16500 bytes/request ≈ 14600 sign. bytes 24 bytes/response	26 bytes/response
	`anon.`	0.016 secs/request 2755 bytes/request 2641 sign. bytes 26 bytes/response	≈ 0.099 secs/request ≈ 30225 bytes/request ≈ 24423 sign. bytes 24 bytes/response	26 bytes/response
CPY06		**rspStatus**		
		`evidence`	`denied`	`accepted`
reqType	`unlink.`	0.02 secs/request 890 bytes/request 856 sign. bytes 27 bytes/response	≈ 0.09 secs/request ≈ 5290 bytes/request ≈ 5136 sign. bytes 24 bytes/response	26 bytes/response
	`anon.`	0.02 secs/request 886 bytes/request 856 sign. bytes 26 bytes/response	≈ 0.16 secs/request ≈ 9687 bytes/request ≈ 9416 sign. bytes 24 bytes/response	26 bytes/response

Table 6.3: Experimental results using KTY04 (top) and CPY06 (bottom) obtained for ACFP requests of type `unlinkability` and `anonymity` for cases where no evidence is sent (column `evidence`, where no evidence at all is sent), evidence not satisfying the policy in Listing Figure 6.17 (column `denied`, where 5 group signatures are sent), and evidence satisfying the policy (column `accepted`, where 10 group signatures are sent). All values are averaged over 1000 requests/responses.

6.6 Chapter conclusion

In this chapter, we have outlined the necessity of the definition of standard friendly mechanisms for the management of fair anonymity systems. The need of such auxiliary but essential functionality stems from the fact that without practical means for revoking anonymity, it will keep generating mistrust between service providers. These protocols actually allow to incorporate the already existing cryptographic approaches for fair anonymity, providing a means to reach an equilibrium in the balance between the right of users to privately access information and the right of service providers to prevent misuses of their services.

Moreover, with the extensions herein proposed to the X.509 for granting *unlinkability* and *anonymity* revocations for *anonymous certificates*, we actually reach the same functionality than was already available for normal X.509 certificates in a privacy respectful manner that does not threaten legitimate users of anonymity. Further, we also

create the necessary protocol (ACFP) for conveying any information that may be required to back up a revocation request. Instead of incorporating the functionality provided by ACFP into OCSP, we have chosen to create a new X.509-like protocol in order to maintain a separation of each different functionality, for the sake of simplicity and modularity. The combination of these new mechanisms enables the implementation of *fair anonymity* via the X.509 infrastructure.

Moreover, the proposal of the mentioned mechanisms is accompanied by some experimental results via a prototype, showing its feasibility. This provides actual data showing that the already proposed cryptosystems backing up the fairness functionality, when implemented through our proposals, incur in bearable additional costs. Moreover, since our mechanisms are independent on the underlying cryptographic scheme, switching between the existing alternatives is straightforward.

Regarding possible future work, an straightforward option is to implement these protocols under the X.509 standard. This would allow them to actually be deployed in current infrastructures, and get more insight of their behavior and limitations. Also, we emphasize again that the three mechanisms (the extensions to CRL and OCSP, ACFP) allow further extensions in order to incorporate additional functionality, may the need arise. In fact, during the peer-review process of the work in [84], the extension of *delta* OCSPs (see Section 6.2.2) was suggested by a reviewer. Another possible extension could be to differentiate what kind of privacy property has been revoked, i.e., instead than just returning a `revoked` status, respond with additional `anonRevoked` and `unlinkRevoked` literals. However, this may not be desired in some situations (e.g., provide this information only to authenticated clients). Additionally, if the costs of the extensions to OCSP are considered unacceptable, exploring the compatibility with the OCSP stapling extension [162] may constitute a good option. Through OCSP stapling, the owner of the certificate (usually, a server) has to attach an OCSP response, signed by the appropriate CA, to the message sent during the establishment of a connection. This response (which may be fetched in advance) proves that the associated certificate was valid at some recent time. Therefore, the latency time of an authentication is reduced through "precomputation". The contributions of the community (for instance, via an RFC) would undoubtedly provide invaluable insight for any desirable extension. Finally, the key further action would be to actually implement the proposed extensions and protocols, in order to enable the functionality they provide in real systems.

With the help of anonymity management mechanisms like the ones presented in this chapter, fairness is actually possible in such systems using standard infrastructures which greatly eases their deployment. In Chapter 7, we propose a complete online shopping solution where privacy provided through fair anonymity takes a central role. Furthermore, Chapter 8 defines an extension to the Tor network based on anonymous identities. The identities in both systems could actually be managed with this extensions, and implemented through the library in Chapter 4.

Part IV

Systems

Finally, in Chapter 7 of Part IV we compose a complete e-commerce system that has been designed with privacy in mind. It is based on the advanced cryptographic primitives introduced in Section 2.4. We also apply the security design methodology of Part II in order to check that it meets the required security properties. Moreover, with the help of the group signatures library also described in Part II, we show the results of an experimental analysis we have performed to check that it does not incur in unacceptable additional costs. Additionally, in Chapter 8 we outline an extension for incorporating fairness into the Tor network. Both the e-commerce system and the Tor extension are compatible with the registration protocol in Chapter 5 in order to distribute robust cryptographic anonymous identities; and with the mechanisms and protocols for anonymity management described in Chapter 6.

CHAPTER 7

Caduceus: comprehensive fair anonymous online shopping

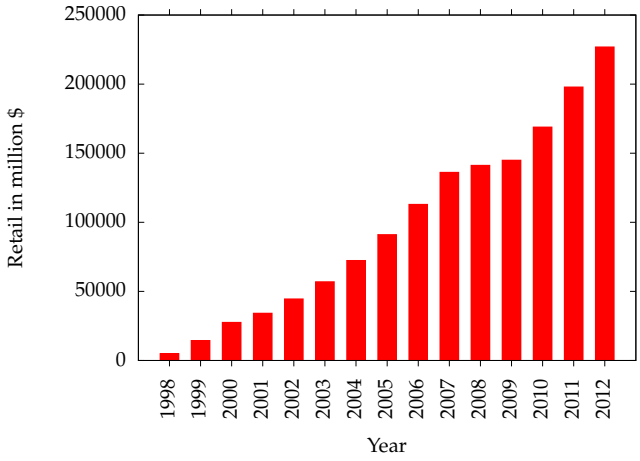
Chapter based on and supported by references [87].

E-commerce, and accordingly online shopping, has been growing continuously (see Figure 7.1), most likely due to the availability and convenience that it has brought to consumers. Companies also enjoy e-commerce as they can improve sales by collecting useful data from consumers, so as to serve consumers better and obtain greater profits.

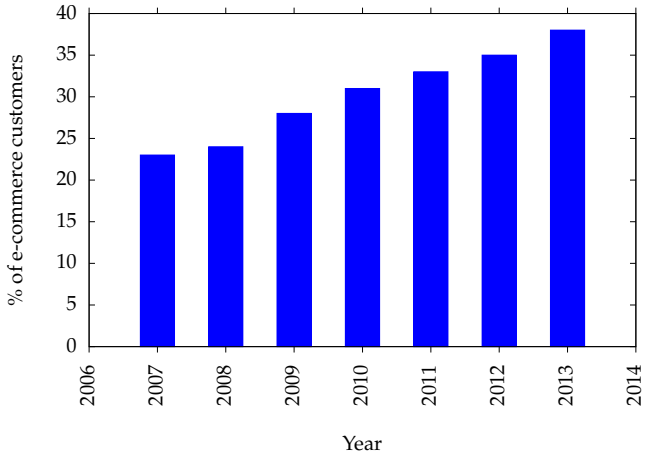
However, at times, consumers may feel uneasy with these practices since user data may be misused and thus may hurt aspects of their personal privacy [68, 72, 165]. Consequently, a central question is whether a privacy supporting solution in this domain is possible. A central challenge in this scenario is to find a good balance point so that not only consumers but also companies (and other participating parties) find the system useful. Considering that e-commerce mechanisms have to deal with various important aspects of sales and payment processes, it is almost certain that any privacy-preserving mechanism with poor support of some of these crucial features or processes would simply be rejected in the real world.

In this chapter, we design a practical online shopping mechanism that provides consumers with a viable choice to protect their privacy (when desired). Our proposal constitutes a comprehensive system, meaning that it provides a functionality set equivalent to that of currently deployed systems. Moreover, it does so while respecting the privacy of its users. Furthermore, we provide some experimental results obtained by means of a prototype of our proposal, showing that the additional costs are actually reasonable. As mentioned in previous chapters, this system could directly benefit from a variant of the registration protocol described in Chapter 5 and the anonymity management mechanisms introduced in Chapter 6.

Moreover, the security of this system has been verified using the methodology in Chapter 3. Specifically, during the procedural verification stage this verification includes the formal and computational approaches. In this case, this is specially important since the system is composed by complex cryptographic schemes and several communication protocols. The formal verification assumes perfect cryptographic primitives, and thus tries to *break* the system by automatically trying all the possibilities of message alteration, replays and/or impersonation (by creating new "illegitimate" messages). The computational verification (done in [87]), on the other hand, deals with the security of the system given the employed cryptographic primitives, and does not assume that they are unbreakable. Instead, the security proofs are built on the grounds



(a) E-commerce retail sales (in million dollars) in USA between 1998 and 2012. Data source [155].



(b) Percent of users having bought via e-commerce in the last 3 months in EU-28. Data source [99].

Figure 7.1: Indicators of e-commerce growth in USA and EU-28.

of the properties that have been mathematically proved for the cryptographic blocks that serve as a basis for our system. Thus, as we stated in Chapter 3, this combination offers further guarantees that our system provides the claimed security properties. Actually the achieved *Protocol Assurance Level*, as defined in [142], would be the maximum, PAL4. We have called this system Caduceus, after the homonymous staff carried by Hermes, the greek god for trade and commerce.

7.1 Related work

The advantages that online shopping has brought to customers and retailers are straight forward. To the former it has brought comfort, while to the latter it allows to increase their target market and reduce costs. However, e-commerce has also brought the benefits and drawbacks of big data. As studied in [72], payment information (specifically, credit card metadata) is highly sensitive, allowing to easily identify customers from pseudonymized data sets, requiring very few additional information. Moreover, this is worsened by the circumstance that payment details may also be directly leaked from merchants to payment providers and from payment providers to other data aggregators. While the minimum requirement for payment orders is to share the order total, receiving merchant and authenticated payment method, merchants seem to share much more additional information, like products details, names, shipping costs and even the customers' names and addresses [165]. Thus, [72] states that few additional information is necessary in order to reidentify customers from pseudonymized metadata sets, and [165] shows that many times, this additional information may be leaked. This certainly calls for online shopping and e-commerce systems that follow the privacy-by-design principle, minimizing the risks of these threats as much as possible.

Privacy respectful online shopping has been divided in two types of systems depending on the information hidden to the service providers [169]: private purchase systems hiding the purchased items; and anonymous purchase systems, hiding buyers' identities. In [169], private purchase protocols are endorsed over anonymous purchase ones, arguing that they allow customer management. These systems are based on Priced Oblivious Transfer [10], and their anonymous access control mechanisms are usually implemented through anonymous credentials [53, 69]. Also, according to [169], anonymous purchase systems are incompatible with payment methods requiring buyers' authentication. In this case, they are usually based on e-cash [64], a powerful primitive that has originated an extensive research in the context of payment systems [58, 71, 152, 171], although real systems using it are yet scarce. In [125] an account-based anonymous payment system is built upon mix networks (see Definition 8 for *mix networks*), but it introduces extensive changes in the infrastructure. It is also significant the work on reputation systems [19] and signatures of reputation [33]. However, these proposals only support monotonic aggregation of reputation, making it unsuitable when the reputation may be modified in any direction. There is also some previous work about *fair payment systems* [56, 176]. However, as opposed to these e-cash based schemes, ours is account-based, making it suitable for current infrastructures, natively includes fraud prevention and marketing techniques, and allows the financial entities to work with real identities.

7.2 Proposal overview

As stated, our system aims to provide an initial step towards privacy respectful alternatives to currently deployed and widely used e-commerce systems. As such, it must incorporate functionality that may be comparable to them. For that matter, our system incorporates the following features:

- Users opt-in to levels of privacy. Our system lets customers choose whether to use anonymity for each transaction. Anonymous transactions are not linked to the customer. Non-anonymous transactions (which are actually pseudonymous by

design) update customers' history so that they may get better deals in the future (but this is the customer choice).

- Marketing tools. Our system supports marketing promotion/ affinity mechanisms such as coupons, that are available for customers independently on whether they purchase anonymously or not.
- Fraud prevention filters. Our system supports an appropriate set of tools to filter a large set of fraud attempts from malicious customers, in a similar manner than it is currently done in worldwide deployed systems.
- Dispute resolving. Our system has a procedure to resolve disagreements between customers and merchants after a customer has paid for a product.
- Anonymous delivery. Our system is consistent with an anonymous delivery system [18], and thereby suitable for both digital and physical goods.

As for the entities taking part in the system, in addition to the necessary customers and merchants, we define two special parties: the payment system (PS), and the financial network (FN). PS is the mediator of every transaction: customers first pay for the products to PS, which later settles the debt with merchants. Moreover, PS is responsible for providing customers with marketing promotions, performing fraud prevention procedures, contacting with FN in order to request credit/debit card validation and completing checkout orders. FN is an abstraction of the Acquirer Bank, Issuer Bank and Card Associations bundled together. We assume that PS and FN are semi-honest, i.e. they follow the protocol (since they model entities which are regulated and want to stay in business).

It is worth to emphasize the fact that these four entities are precisely the same entities that intervene in typical online shopping systems.

7.2.1 Comparison with industry systems

In existing systems, a transaction starts when a customer C browses the online website of some merchant M, fills up his shopping cart, and finally clicks on a checkout button. Then M forwards the relevant purchase information to a Payment System (PS) such as Amazon, Google Checkout and PayPal; PS processes it, applies marketing and fraud prevention techniques (some of which may be applied by M instead), and contacts the Card Network (for card-based payments) for verifying that C has provided valid payment information and has enough funds. If these checks are successful, M proceeds to deliver the purchased goods. Simultaneously, PS, through the Card Network, instructs M's bank to reimburse M. In turn, M's bank requests a reimbursement from C's bank, who bills C accordingly. Finally, if PS acted as intermediary for the payment, it eventually settles the debt with M. This process, which may vary slightly depending on the context, is depicted in Figure 7.2. A more detailed explanation is available at [27, pp. 94–97].

We stress that *our system essentially respects the overall infrastructure, information flow, and entities described above* except we simplify the model by bundling up all financial entities (Credit Card Network and banks) in a single entity, the Financial Network (FN); this simplification would not affect security, since we are mainly concerned about customer privacy against merchants, and we will assume that financial entities are honest-but-curious. Moreover, while we assume card-based payments, this abstraction allows us to easily adapt our system to alternative payment methods, as outlined in Section

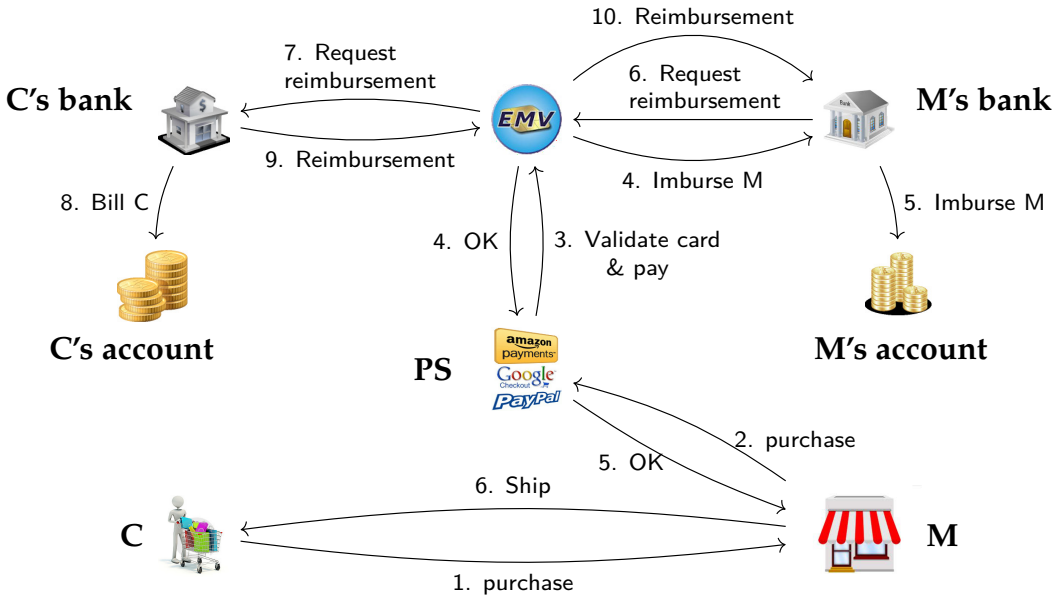


Figure 7.2: Payment process infrastructure and information flow.

7.4. As before, a transaction starts when Customers (C) interact online with merchants (M), who forward orders to the Payment System (PS). PS performs certain fraud prevention and also marketing techniques, which makes sense, since it is PS who has complete visibility over every user actions with all the merchants. However, it is possible (as in current systems) to shift part of this load to merchants. For instance, M may choose not to sell a product worth more than $1000 to customers in foreign countries (fraud prevention) or grant $10 discounts to local customers for purchases in the local store (marketing).

7.2.2 Building blocks

We make use of two important cryptographic primitives to improve privacy: group signatures and partially blind signatures. The combination of these two primitives enables the creation of privacy respectful but traceable (when needed) tokens of information that allow users to perform purchases as if they were using a conventional system. Both primitives, as well as additional ones that are also employed here, were described in Section 2.4, along with the notation employed here for interacting with them.

7.2.3 Functionality

In this section, we introduce the main functionality of the system. In short, the overall process may be divided in an initial setup stage in which every party performs the computations required to obtain the cryptographic tokens and a subsequent stage in which all the operations typical to an online shopping setting are run. In the latter, a customer starts by retrieving a *turn* by interacting pseudonymously with PS. This turn entitles her to complete a checkout, and it is in this phase when the user obtains the marketing promotions she is eligible for. In addition, part of the fraud prevention mechanisms are also run at this point. Subsequently, the checkout phase is initiated, by means of which the user interacts with the merchant, who in turn interacts with PS who addi-

tionally communicates with FN. In this checkout phase, where the previously obtained turn acts as central token for the authentication process, the customer decides whether she wants to act anonymously or pseudonymously. In the former case, the checkout phase is unlinkable from the turn retrieval phase. However, in order to guarantee the unlinkability, PS acts as an intermediary between customer and merchant, initially receiving the payment in stead of the latter[1]. Therefore, an additional settlement phase is required, in which PS performs batch payments to settle the corresponding debts with the merchants. Finally, in the case a dispute arises between any of the entities, a dispute solving process, which also preserves the privacy of the previous phases, may be launched.

In the following paragraphs, we give a more detailed definition of the mentioned stages and the functions that compose them.

System setup. The system entities are initialized as follows:

- $(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \mathtt{FNSetup}(1^k)$. Generate key pairs for FN.
- $(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \mathtt{PSSetup}(1^k)$. Generate key pairs for PS.
- $(pk_{M_j}, sk_{M_j}) \leftarrow \mathtt{MSetup}(1^k)$. Generate a key pair for a merchant M_j.
- $\langle (P_i, mk_i), (P_i, \ell') \rangle \leftarrow \mathtt{CSetup}(pk_{\mathsf{FN}})[C_i(s_i), \mathsf{FN}(sk_{\mathsf{FN}}, \ell)]$. Customer C_i contacts FN, creates an account and obtains a membership key mk_i and pseudonym P_i using a secret s_i. FN updates its membership database ℓ into ℓ'.

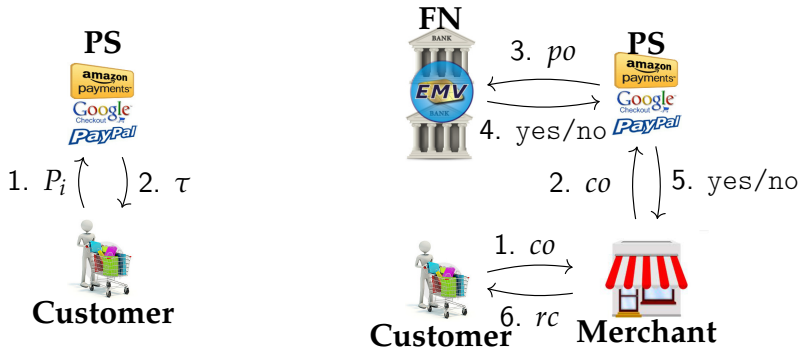


Figure 7.3: *Turn-retrieval* (left) and *Checkout* (right) phases.

Turn-retrieval. In this phase, a customer contacts PS and retrieves a turn for the subsequent checkout action.

- $\tau \leftarrow \mathtt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P_i)[C_i(mk_i), \mathsf{PS}(sk_{\mathsf{PS}})]$. A customer C_i authenticates with PS using P_i, and obtains a turn τ. The turn includes information of marketing promotion, risks, and deadline. We denote those pieces of information by $\tau.pr$, $\tau.rk$, and $\tau.dl$. Figure 7.3 sketches this process.

The system provides the turn verification algorithm as well: $\mathtt{VerifyTurn}(pk_{\mathsf{PS}}, \tau)$.

[1]Note that this is the usual behavior of current Payment Systems.

Checkout. After receiving the turn, the customer completes the purchase by running a pseudonymous (*IssueCheckout*) or anonymous (*IssueAnonCheckout*) procedure. Let α be the product information including the product name, selling merchant, and so on; let \$ be the price of the product. In addition, let β be the customer's billing information including the customer's name, credit card number, card security number, billing address, etc; moreover, β contains a random number to uniquely identify each transaction of the customer. The checkout process is decomposed into the following algorithms. For readability, we omit the public keys from their inputs.

- $co \leftarrow \texttt{IssueCheckout}(mk_i, P_i, \tau, \alpha, \$, \beta)$. C_i, with its private key mk_i, runs this algorithm to generate checkout information to be sent to the merchant M_j. (The merchant will pass the checkout co to the payment system PS.)
- $co \leftarrow \texttt{IssueAnonCheckout}(mk_i, \tau, \alpha, \$, \beta)$. The customer runs this algorithm for anonymous checkout.
- $po \leftarrow \texttt{IssuePmtOrder}(sk_{\mathsf{PS}}, co)$. The payment system PS runs this algorithm to generate a payment order to be sent to FN. (Then, FN checks if the order is valid, and if so, FN will process the order and let PS know when the process is finished; in turn, PS will let the merchant know that the checkout process is successfully completed.)
- $rc \leftarrow \texttt{IssueReceipt}(sk_{M_j}, co)$. When the checkout is successfully completed, merchant M_j runs this algorithm to generate a receipt to be sent back to C_i.

In order to check the validity of the various pieces of information above, the system supports public verification algorithms:
$\texttt{VerifyCheckout}(co), \texttt{VerifyPmtOrder}(po), \texttt{VerifyReceipt}(rc, co)$.

Settlement and dispute resolution. PS periodically settles debts with merchants. Disputes are resolved by the client showing a receipt (or a checkout) and claiming its ownership.

7.2.4 A sample scenario

In order to consolidate the previous processes, we shortly give now a high level overview of how a purchase process would be in a real world scenario. The description begins at the moment in which a customer starts browsing some merchant's website and ends when she receives the purchased goods. We assume that all parties have the required keys.

1. Initially, a customer C accesses Merchant's M website. C produces a shopping cart.
2. C then clicks a "Checkout now" button, which redirects her to a web service (or equivalent) controlled by PS.
3. Within this session between C and PS, the process `TurnRetrieval` is run imperceptibly to the customer, perhaps excluding the need to specify the keys to use (also, if C is eligible for any marketing promotion, a selection menu may appear). As a result of this process, C obtains a token (the turn) that:

- Entitles him for performing exactly one checkout.
- Is linked to a specific risk estimation associated to the customer, which would serve the merchant as a fraud prevention mechanism.
- It includes the marketing promotions that the customer has chosen.
- Is only valid for a predefined time interval.
- Furthermore, the previous properties cannot be modified once the turn has been issued.

Finally, note that steps 2 and 3 may be exchanged in order, thus allowing *C* to pick up promotions before filling the shopping cart.

4. After `TurnRetrieval` completion, the connection between *C* and *PS* is closed and the `DoCheckout` process between *C* and *M* starts. Possibly, some random delay may be introduced, to avoid time-based deanonymization.

5. At this point, *C* is asked whether to run the checkout process anonymously and pseudonymously, and introduces the payment information. This choice has the following consequences:

   - In a pseudonymous checkout, the customer purchase history is updated based on the pseudonym, which will be entitled for better marketing promotions in the future.
   - In an anonymous checkout, the customer purchase history is not updated (since there is no way to link the purchase to the customer), but he will not receive better marketing promotions in future purchases.

6. After making this decision, all the subprocesses included within `DoCheckout` are run.

7. If `DoCheckout` succeeds, *C* obtains the checkout receipt. *M* ships or sends the purchased goods and *PS* updates the settlement database.

8. In case of dispute, *C* can use the receipt to prove, anonymously, that he has actually paid for the purchased goods.

7.2.5 System requirements

We assume that all the communications in which a customer is involved are performed in a sender-anonymous channel such as Tor [91]. The application level communications are supposed to be layered on top of some security protocol providing confidentiality such as TLS [89]. When customers join the system, they receive a smart card containing their private keys in a secure manner[2]. For legal reasons, we assume that the financial network FN needs to know the real identities of the parties involved in a transaction.

As for the entities assumptions, we start from the consideration that customers and merchants can collude with everyone except FN. PS acts as a semi-honest party, but can collude with everyone except FN. FN is semi-honest and cannot collude with any other entity.

[2]The keys and infrastructure could be stored and managed as proposed in [31, 78]

Entities			
C_i	A customer, with member key mk_i	$\mathtt{M}_j$	A merchant
PS	The Payment System	FN	The Financial Network
Information tokens			
rk	Account risk estimation	pr	Promotions set
dl	Turn deadline	τ	Turn
\$	Purchase price	α	Purchase information
β	Billing information	$\tilde{\beta}$	Billing information encrypted with pk_{FN}
ϱ_α	Group signature of α	$\varrho_{\tilde{\beta}}$	Group signature of $\tilde{\beta}$
ψ	ZK proof for τ,ϱ_α and $\varrho_{\tilde{\beta}}$	co	Checkout order $= (\tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
po	Payment Order $= (\$, \tilde{\beta}, \mathtt{AVS}, \varrho_{\tilde{\beta}})$	rc	Receipt
AVS	Address Verification System		

Table 7.1: Summary of symbols.

7.3 System description

We now give a construction of the system outlined in Section 7.2.3 using the building blocks of Section 7.2.2. For this construction, we use the notation described in the paragraph *Notation for cryptographic protocols* in Section 2.2. Additionally, and for readability, we sometimes assume that if any internal step fails, the overall process also fails and stops. In order to serve as a quick reference summary, Table 7.1 contains the identifiers employed to denote each entity, and all the miscellaneous symbols that are used in the following definitions.

7.3.1 System setup

FN sets up the master key for group signatures and generates a public-key encryption keypair. PS generates the keypairs for issuing partially blind signatures and digital signatures. Each merchant M_j generates a keypair for digital signatures. Customers join a group G by initiating a two-party process with FN. Customer C_i specifies some secret s_i and obtains his private member key mk_i for group G. The pseudonym P_i is a group signature on a random message created using his membership key mk_i; we let $P_i.r$ denote the random message and $P_i.\varrho$ the group signature on $P_i.r$. Refer to Figure 7.4 for the detailed procedures.

7.3.2 Turn-retrieval

The turn-retrieval phase is a two-party protocol between a customer and PS. The protocol starts by having the customer C_i send its pseudonym P_i. Then, PS retrieves the information of how loyal this customer is (i.e., rk), whether (and how) the customer is eligible for marketing promotions (i.e., pr), and the deadline of the turn to be issued (i.e., dl), sends back $(\mathtt{rk}, \mathtt{pr}, \mathtt{dl})$ to C_i. C_i chooses a subset $\mathtt{pr}'$ from the eligible marketing promotions pr. Finally, C_i will have PS create a partially blind signature such that its common message is $(\mathtt{rk}, \mathtt{pr}', \mathtt{dl})$ and its blinded message is a commitment com to its membership key mk_i. We stress that the private member key mk_i of the customer C_i links the pseudonym (i.e., $\varrho_{P_i} = \mathtt{GS.Sign}_{mk_i}(P_i)$) and the blinded message (i.e., $\mathsf{com} = \mathsf{Com}(mk_i, r_{com})$). The customer is supposed to create a ZK-PoK ϕ showing this link. Upon successful execution, the turn is set to τ. We use τ.rk, τ, pr, τ, dl, τ.com, $\tau.\varrho$

$((pk_{FN}, pk_G), (sk_{FN}, sk_G)) \leftarrow \texttt{FNSetup}(1^k)$
$(pk_G, sk_G) \leftarrow \texttt{GS.Setup}(1^k)$
$(pk_{FN}, sk_{FN}) \leftarrow \texttt{Gen}(1^k)$

$((pk_{PS}, pk_{PBS}), (sk_{PS}, pk_{PBS})) \leftarrow \texttt{PSSetup}(1^k)$
$(pk_{PBS}, sk_{PBS}) \leftarrow \texttt{PBS.KeyGen}(1^k)$
$(pk_{PS}, sk_{PS}) \leftarrow \texttt{SGen}(1^k)$

$\langle (P_i, mk_i), (P_i, \ell') \rangle \leftarrow \texttt{CSetup}(pk_G)[C_i(s_i), \mathsf{FN}(sk_G, \ell)]$
$\langle mk_i, \ell' \rangle \leftarrow \texttt{GS.Join}(pk_G)[C_i(s_i), \mathsf{FN}(sk_G)]$
C_i chooses $r \leftarrow \{0,1\}^*$
C_i computes $\varrho \leftarrow \texttt{GS.Sign}_{mk_i}(P_i, mk_i; s_{P_i})$
C_i sends $P_i = (r, \varrho)$ to FN

$(pk_{M_j}, sk_{M_j}) \leftarrow \texttt{MSetup}(1^k)$
$(pk_{M_j}, sk_{M_j}) \leftarrow \texttt{SGen}(1^k)$.

Figure 7.4: Setup procedures.

to denote the risk factor, marketing promotions, deadline, commitment to the member key, and the resulting blind signature respectively. Refer to Figure 7.5 for pictorial description. Turns are verified using the `VerifyTurn` process, defined as follows:

$$\texttt{VerifyTurn}(pk_{\mathsf{PS}}, \tau) : \textbf{return } \texttt{PBS.Verify}(\tau.\varrho, (\tau.\texttt{pr}, \tau.\texttt{rk}, \tau.\texttt{dl}), \tau.\mathsf{com}, pk_{\mathsf{PS}})$$

C_i		PS
	P_i →	
		Verify the signature $P_i.\varrho$ on $P_i.r$. If verification fails, abort. rk : account risk for P_i pr : promotions for P_i dl : turn deadline
	← $(\texttt{rk}, \texttt{pr}, \texttt{dl})$	
$\mathsf{com} \leftarrow \mathsf{Com}(mk_i; r_{\mathsf{com}})$ $(\tilde{\mathsf{com}}, \pi) \leftarrow \texttt{PBS.Blind}(\mathsf{com}; s)$ $\phi = \texttt{ProveZK}(x; w)$ where $x = (\tilde{\mathsf{com}}, \pi, P_i, \varrho_{P_i}, pk_G), w = (mk_i, r_{\mathsf{com}}, s, s_{P_i})$ such that : $(\tilde{\mathsf{com}}, \pi) = \texttt{PBS.Blind}(\mathsf{Com}(mk_i; r_{\mathsf{com}}); s)$, $\varrho_{P_i} = \texttt{GS.Sign}_{mk_i}(P_i; s_{P_i})$. $\texttt{pr}'$: customer's choice from pr		
	$(\texttt{pr}', \tilde{\mathsf{com}}, \pi, \phi)$ →	
		$\texttt{VerifyZK}(\tilde{\mathsf{com}}, \pi, P_i, \varrho_{P_i}, pk_G)$ If verification fails, abort. $\tilde{\varrho} \leftarrow \texttt{PBS.Sign}_{sk_{PBS}}((\texttt{rk}, \texttt{pr}', \texttt{dl}), \tilde{\mathsf{com}})$
	← $\tilde{\varrho}$	
$\varrho \leftarrow \texttt{PBS.Unblind}_{pk_{PBS}}(\tilde{\varrho}, \tilde{\mathsf{com}}, s)$ **output** $\tau = ((\texttt{rk}, \texttt{pr}', \texttt{dl}), \mathsf{com}, \varrho))$		

Figure 7.5: Turn-retrieval process.

7.3.3 Checkout

In this section, we describe the checkout phase procedures.

Step 1: Client issues a checkout object. A customer C_i enters the checkout phase by creating a checkout object *co* by executing `Issue(Anon)Checkout` using the turn τ obtained during the turn-retrieval phase. In either procedure, C_i generates two group signatures: one signature ϱ_α on α, the other $\varrho_{\tilde{\beta}}$ on \$ and $\tilde{\beta}$. Then, C_i generates a ZK proof ψ showing that the turn and the group signatures (and the pseudonym for pseudonymous checkout) use the same member key mk_i. In summary, we have $co = (\tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$.

$co \leftarrow$ `IssueCheckout`$(mk_i, P_i, \tau, \alpha, \$, \beta)$:
- $\varrho_\alpha \leftarrow \texttt{GS.Sign}_{mk_i}(\alpha; r_\alpha)$
- $\tilde{\beta} \leftarrow \texttt{Enc}_{pk_{FN}}(\beta)$
- $\varrho_{\tilde{\beta}} \leftarrow \texttt{GS.Sign}_{mk_i}((\$, \tilde{\beta}); r_\beta)$
- $\psi \leftarrow \texttt{ShowZK}(x, w)$ with
  - $x = (P_i, \tau.\mathsf{com}, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}})$
  - $w = (s_{P_i}, mk_i, r_{\mathsf{com}}, r_\alpha, r_\beta)$
  - such that
    - $P_i.\varrho = \texttt{GS.Sign}_{mk_i}(P_i.r; s_{P_i})$
    - $\tau.\mathsf{com} = \mathsf{Com}(mk_i; r_{\mathsf{com}})$
    - $\varrho_\alpha = \texttt{GS.Sign}_{mk_i}(\alpha; r_\alpha)$
    - $\varrho_{\tilde{\beta}} = \texttt{GS.Sign}_{mk_i}((\$, \tilde{\beta}); r_\beta)$
- $co \leftarrow (P_i, \tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
- **return** *co*

$co \leftarrow$ `IssueAnonCheckout`$(mk_i, \tau, \alpha, \$, \beta)$:
- $\varrho_\alpha \leftarrow \texttt{GS.Sign}_{mk_i}(\alpha; r_\alpha)$
- $\tilde{\beta} \leftarrow \texttt{Enc}_{pk_{FN}}(\beta)$
- $\varrho_{\tilde{\beta}} \leftarrow \texttt{GS.Sign}_{mk_i}((\$, \tilde{\beta}); r_\beta)$
- $\psi \leftarrow \texttt{ShowZK}(x, w)$ with
  - $x = (\tau.\mathsf{com}, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}})$
  - $w = (s_{P_i}, mk_i, r_{\mathsf{com}}, r_\alpha, r_\beta)$
  - such that
    - $\tau.\mathsf{com} = \mathsf{Com}(mk_i; r_{\mathsf{com}})$
    - $\varrho_\alpha = \texttt{GS.Sign}_{mk_i}(\alpha; r_\alpha)$
    - $\varrho_{\tilde{\beta}} = \texttt{GS.Sign}_{mk_i}((\$, \tilde{\beta}); r_\beta)$
- $co \leftarrow (\tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
- **return** *co*

Figure 7.6: `IssueCheckout` and `IssueAnonCheckout` processes run by customers for initiating the checkout phase pseudonymously or anonymously, respectively.

Step 2: Merchant sets up AVS fraud prevention. When merchant M_j receives the checkout object *co*, M_j verifies *co* by running `VerifyCheckout`. If the verification succeeds, M_j passes this object by attaching an Address Verification Service value (`AVS`) that will be subsequently used as reference for fraud prevention.

`VerifyCheckout`$(co, pk_{\mathsf{PS}}, pk_G)$:
- Parse *co* into $([P_i], \tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
- `VerifyTurn`(τ, pk_{PS})
- Check if $(\tau.\texttt{rk}, \tau.\texttt{pr}, \tau.\texttt{dl})$ is acceptable.
- $\texttt{GS.Verify}_{pk_G}(\varrho_\alpha, \alpha)$
- $\texttt{GS.Verify}_{pk_G}(\varrho_{\tilde{\beta}}, (\$, \tilde{\beta}))$
- $\texttt{VerifyZK}(([P_i], \tau.\mathsf{com}, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}), \psi)$
- If all the checks above pass, **return** 1
- Otherwise return 0

Figure 7.7: `VerifyCheckout` process, run by merchants.

Step 3: PS issues a payment order po. On receiving co and $\texttt{AVS}$ from M_j, PS verifies co, runs $\texttt{IssuePmtOrder}$ and issues a payment order po with the minimum information required by FN for processing the payment that is, $po = (\$, \tilde{\beta}, \varrho_{\tilde{\beta}}, \texttt{AVS})$.

$po \leftarrow \texttt{IssuePmtOrder}(sk_{\mathsf{PS}}, co, \texttt{AVS})$:
$\texttt{VerifyCheckout}(co)$
If verification fails, return 0
Parse co into $([P_i], \tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
$po \leftarrow (\$, \tilde{\beta}, \texttt{AVS}, \varrho_{\tilde{\beta}})$
return po.

Figure 7.8: $\texttt{IssuePmtOrder}$ process, run by PS.

Step 4-5: Payment confirmations. Given the payment order po, FN verifies it by running $\texttt{VerifyPmtOrder}$. If the verification succeeds, FN processes the order and notifies PS of the completion; PS in turn sends the confirmation back to M_j.

$\texttt{VerifyPmtOrder}(po, sk_G, sk_{FN})$:
Parse po into $(\$, \tilde{\beta}, \texttt{AVS}, \varrho_{\tilde{\beta}})$
$\texttt{GS.Verify}_{pk_G}(\varrho_{\tilde{\beta}}, (\$, \tilde{\beta}))$;
$\beta = \texttt{Dec}_{sk_{FN}}(\tilde{\beta})$;
Check if β has not been used before.
Check if $\texttt{GS.Open}_{sk_G}(\varrho_{\tilde{\beta}})$ equals C_i in β.
Verify the other billing information in β.
Check if $\texttt{AVS}$ filter on β has any issues.
If all the checks above pass, return 1
Otherwise return 0

Figure 7.9: $\texttt{VerifyPmtOrder}$ process, run by FN.

Step 6: M_j issues a receipt. When merchant M_j receives the confirmation from PS, it runs $\texttt{IssueReceipt}$, producing a signature on co, signed by M_j. Finally, C_i verifies that this receipt is correct.

$rc \leftarrow \texttt{IssueReceipt}(sk_{M_j}, co)$:
$rc \leftarrow \texttt{Sig}_{sk_{M_j}}(co)$
return rc.

Figure 7.10: $\texttt{IssueReceipt}$ process, run by FN.

7.3.4 Settlement

To avoid FN being able to link customer and merchant, PS becomes the payee during checkout. After a predefined lapse of time (enough to get a volume of transactions

making the subset-problem hard), PS settles its debts with merchants. This delay improves, both, privacy and performance.

7.3.5 Dispute resolution

We define dispute as any disagreement between customers and merchants or PS occurring after a customer has paid. With ShowReceiptZK and VerReceiptZK, defined in Figure 7.11, the customer sends proofs of purchase and payment to a verifier (probably, PS or some M_j). If the customer received a receipt *rc*, he shows *rc* along with the corresponding checkout object *co*; then, using his membership key mk_i, he claims ownership of a group signature contained in *co*. Even if he did not receive a receipt, he can show *co* to PS and claim the ownership of a group signature contained in *co*; regenerating the receipt on-the-fly.

ShowReceiptZK$(co, [rc], mk_i)$:
Parse *co* into $([P_i], \tau, \alpha, \$, \tilde{\beta}, \varrho_\alpha, \varrho_{\tilde{\beta}}, \psi)$
$\pi \leftarrow \text{GS.Claim}_{mk_i}(\varrho_\alpha)$
return $(co, [rc], \pi)$

VerReceiptZK$(co, [rc], \pi)$:
Verify *co* succeeds.
Extract M_j from *co*.
If *rc* exists, $\text{Ver}_{pk_{M_j}}(co)$
$\text{GS.ClaimVerify}_{pk_G}(\pi, \varrho_\alpha)$
If all the checks pass, return 1.
Otherwise return 0.

Figure 7.11: ShowReceiptZK and VerReceiptZK processes for dispute solving.

7.4 Additional functionality

We now give some insights into additional features supported by our system which, despite not having been included within the main description, actually help to make it richer and more flexible. It is actually this additional functionality which, based on the privacy-enhancing design defined in previous sections, makes our system comparable to current ones.

Fraud prevention filters. Our system supports the following fraud prevention filters. Each filter is on the transaction-level (tx-level) or the account-level (acc-level), depending on whether it checks the information specific to transactions or to accounts.

- *Pseudonym velocity filter (acc-level):* It measures how many recent purchases have been initiated by the same pseudonym (similar to PayPal's IP velocity filter). It can be applied during turn-retrieval.
- *Suspicious shipment changes (acc-level):* This filter can be added by making the city/country of shipment visible, e.g., including it in the common message of the partially blind signature obtained during turn-retrieval.
- *Address verification system (AVS, acc-level):* This filter is natively included in our system. It is a verification typically performed by Credit Card companies matching the billing address specified by customers with the billing address associated to a credit card.

- *Billing/Shipping Address Mismatch Filter (tx-level):* With Locality Sensitive Hashing (LSH) [62, 158], this filter can be added as follows. C computes two hashes $\tilde{b} \leftarrow H(b, r)$, $\tilde{s} \leftarrow H(s, r)$, where b and s are the billing and shipping address, respectively, and r is a random value. Then it sends $(r, \tilde{b}, \tilde{s})$ to M during checkout, who compares them. The probability of obtaining a match will be proportional to the similarity of b and s due to the properties of LSH. Since FN later checks if $\tilde{b}$ is equal to $H(billing, r)$ using the actual billing address $billing$, the filter works correctly.

- *Maximum price per order (tx-level):* This filter is trivial to apply by M or PS.

- *Maximum number of items per order (tx-level):* Trivial to apply by M or PS.

- *Other filters:* Filters like *Currency type*, *Bank Identification Number*, *Transaction type* [127] may be directly applied.

Note that the account-level filters (excluding AVS by FN) are applied by PS during the turn-retrieval phase. Thus, anonymous checkouts do not affect their effectiveness. Transaction-level filters may be applied by either PS or merchants. Also, since an account risk estimation is passed to the checkout phase as part of the turn, both account and transaction risks may be considered jointly.

Marketing techniques. In our system, PS issues marketing promotions during turn-retrieval, consulting the pseudonym's history. In [133], promotions are classified depending on for how long they can be applied (limited or unlimited) and the intended recipient (targeted or untargeted). Untargeted and unlimited (UU) marketing promotions are trivial. The other three combinations may be achieved as follows:

- *Targeted and unlimited (TU).* C creates a group signature on the marketing promotion and includes it in the blindly signed message at turn-retrieval. At checkout, C includes this group signature in the ZK proofs (VerifyZK in the VerifyCheckout process above).

- *Untargeted and limited (UL).* Simply include a deadline in the marketing promotion.

- *Targeted and limited (TL).* Add a group signature as in TU promotions, specifying the target and deadline.

Our system also supports merchant-issued marketing promotions. To do this, a group of merchants should be set up, and each merchant $\mathtt{M}_j$ should have a policy $\mathtt{pr}_{\mathtt{M}_j}$ for marketing promotions issuance. Before initiating turn-retrieval, $\mathtt{M}_j$ would send $(\mathtt{pr}_{\mathtt{M}_j}, \mathtt{Sig}_{sk_{\mathtt{M}_j}}(\alpha))$ to customer $\mathtt{C}_i$. $\mathtt{C}_i$ would then include $\mathtt{pr}_{\mathtt{M}_j}$ within the common message and $\mathtt{Sig}_{sk_{\mathtt{M}_j}}(\alpha)$ within the blinded message, both to be signed by PS during turn-retrieval. After verifying the group signature, PS would just grant the marketing promotions that $\mathtt{C}_i$ is eligible for, based on $\mathtt{pr}_{\mathtt{M}_j}$. During checkout, $\mathtt{M}_j$ would also verify $\mathtt{Sig}_{sk_{\mathtt{M}_j}}$ in order to prevent $\mathtt{C}_i$ using $\mathtt{pr}_{\mathtt{M}_j}$ with another $\mathtt{M}_i$.

Anonymous delivery of physical goods. The proposed online shopping solution directly supports purchasing digital goods (e.g. merchants can send the purchased good along with the receipt). However, it is also desirable to enable it for handling physical goods. In this respect, despite all the data anonymization, if the physical address of the customer is sent to merchants or PS, anonymity is completely broken. A solution is proposed in [18] offering precisely this functionality by creating APOD, implementing a *physical mix network*. In [18], a central authority, named APODA, creates a group for Delivery Companies (DCs) for issuing blind group signatures, and a group for Mail Stations (MSs), who issue conventional group signatures. Each DC creates a group of merchants with which it collaborates. Customers are identified pseudonymously when interacting with merchants, and anonymously and unlinkably with respect to their pseudonym with DCs, MSs and the APODA. The integration of our online shopping solution with APOD could be easily performed. In APOD, the first interaction occurs between customers and merchants, where the former authenticate pseudonymously to the latter. In our system, this occurs naturally during checkout, where customers are authenticated either anonymously or pseudonymously, according to their own choice[3]. Hence, after completing a successful checkout, customers and merchants can initiate the APOD protocol straight away. The rest of the APOD protocol can be applied without modifications. Alternatively, in order to ease the deployment costs on the merchants' side, merchants could shift this functionality to PS (who would just have to forward the received tokens to the merchant).

Contact customer. This might be necessary in extraordinary situations. For pseudonymous checkouts, a pseudonymous email address associated with the customer's account may be used. For anonymous checkouts, one-time email addresses should be employed, e.g. using an anonymous email service, like Bitmessage.ch[4]. Alternatively, customers could prove in ZK ownership of a receipt, and then receive any notification related to it. However, this requires an active behavior from customers instead of just receiving an email.

Customer opinions. In order to add an opinion about a specific purchase, the customer may just generate a group-signed opinion and create a ZK proof (similar to those sent during checkout) covering this group signature and a claim (like in Figure 7.11) of the receipt obtained during checkout. Any valid opinion would then just be shown in the corresponding website.

Subscriptions. If fees are paid in advance, this is easily solved. For physical goods (e.g., using APOD), customers may initially set up multiple shipment information, one for each delivery. For periodic digital goods, M may request C to claim ownership of the checkout group signatures as in Figure 7.11 (adding a ZK proof with the same member key depending on some varying value for preventing pre-computations) to send the periodic digital good.

For recurring payments, the associated instruction could be sent to FN within the payment information. Also, if customer consent is required to renew the subscription, a notification could be sent to him (see *Contact customer* above).

[3] Allowing anonymity (besides pseudonymity) does not conflict with APOD.

[4] `https://bitmessage.ch`. Last access on March 31st, 2015.

Refunds. Customers may make use of the dispute solving mechanisms to prove in ZK that they have paid some good (see Figure 7.11). If the proof succeeds, a refund order may be placed to FN via PS.

Taxation. Assuming that revealing the city, province, state or country of shipment is not privacy threatening, and including it within the common message of the turn, our system provides all the typically necessary information for tax calculation by either PS or merchant. That is, customer (destination) locality, merchant (source) locality and type of good.

Alternative payment methods. We have focused our explanation in credit card-based payments, which are the most frequent in the real world. However, our framework is also compatible with alternative payment methods such as Bitcoin [88, 151] and Ripple[5]. Given the popularity that crypto-currencies and alternative payment methods have gained recently, it seems reasonable to perform a preliminary analysis of how would our online shopping system behave under this new type of payments model.

Note that, since we have abstracted out the details of the payment method through our FN entity, all major modifications derived from a modification in the payment method will be restrained to this entity. Actually, the other entities will just be affected in that they will have to handle different types of payment information, but the information flow and processes will remain exactly the same.

In the specific cases of Bitcoin and Ripple, the entity FN would actually see its role is reduced, mainly acting as an entry point to the payment system. In particular, the payment information that FN may have in its database related to each customer, may not include the real identity of a client, or the billing address. Thus, FN will not be able to compare the identity of the issuer of the received group signature with the one stored in its database; instead, in Bitcoin and Ripple, clients will indirectly prove ownership of an account (represented by a public key) by demonstrating that they control the corresponding private key.

However, group signatures are still useful for enabling privacy respectful fraud prevention and marketing techniques during the rest of the process. Additionally, in Bitcoin/Ripple, the balance of each account is public given the account identifier, and verifying that the customer has enough funds is straightforward. Still, due to lack of certain pieces of information (e.g., billing addresses), some FN-owned filters for verifying other aspects (e.g., the AVS test) may not be applicable. Nevertheless, merchants not willing to accept alternative payment methods may just reject them by using the *Bank Identification Number* filter (Bitcoin/Ripple "acting" as banks).

Combined with Ripple, our system will have greater payment flexibility, while still maintaining good privacy guarantees. With Bitcoin, the degree of trust placed in FN can be greatly reduced, due to the nature of Bitcoin guaranteeing verifiable payment transactions. Moreover, Bitcoin itself does not guarantee anonymity nor unlinkability robustly [20], so the privacy guarantees in our system are beneficial for improving Bitcoin's since, in our system, it is not possible by design to link merchant and customer. Finally, both Bitcoin and Ripple would benefit from the rich set of useful features mentioned above, as this allows them to be native part of a comprehensive online shopping system.

[5] `https://ripple.com/` (accessed on March 31st, 2015). Ripple is an open system for interoperation between different payment methods, e.g., Bitcoin, real currencies, or account-based transactions.

7.5 Security requirements

In this section, we informally state the security requirements of our proposal. A formal verification of these properties and detailed proofs in the computational model are included in Appendix B.

The security of our system is based on the following properties. Recall that we make the assumptions stated in Section 7.2.5.

Correctness. Suppose every party behaves honestly. If a customer obtains a turn and then runs checkout with a merchant using the turn, then the customer will receive the purchased good(s) and the receipt proving the checkout; at the same time he will be charged the corresponding amount.

Privacy. The system possesses the following customer protecting requirements.

- *Customer anonymity*. If a customer creates an anonymous checkout *co*, then no coalition of merchants, PS, and other customers should be able to determine the identity or pseudonym of the customer from *co*.
- *Unlinkable turn-retrieval and checkout*. If a customer creates an anonymous checkout *co*, then no coalition of merchants, PS, and other customers should be able to link *co* to the corresponding turn-retrieval procedure.
- *Transaction privacy against* FN. The financial network FN should not be able to determine the detail of a customer's transaction beyond what is necessary, i.e., the customer identity and the amount of payment; in particular, the product information of each transaction should be hidden from FN.

Unforgeability. The system ensures the following robustness properties.

- *Turn unforgeability*. A customer should not be able to forge a valid turn that contains a risk factor or a marketing promotion or a deadline set by his own choice.
- *Checkout unforgeability.* When C_i receives a turn from PS, it cannot be used by other customer C_j to create a valid checkout *co*, even if they collude.
- *Receipt unforgeability.* No coalition of customers, merchants (other than the target merchant M), and PS should be able to forge a valid receipt that looks originating from M.
- *Receipt claimability.* For any valid receipt issued to an uncorrupted customer, no other customer should succeed in claiming ownership of the receipt.

7.6 Practical aspects

In this section, we first discuss how could the previous proposals made in this thesis help in the actual deployment of Caduceus. Subsequently, we will present some preliminary results that we have obtained through a prototype of our system.

7.6.1 Identity distribution and anonymity management

It stands out that group signatures are the cornerstone of the privacy properties of our system. Therefore, in order for our proposal to be practical and deployable in real systems, these tokens need to be compatible with the existing infrastructure. This is actually what was achieved in Chapter 6, by extending the previous work on X.509 anonymous certificates [31] in order to support anonymity and unlinkability revocation.

Consequently, the group member's keys should be distributed in X.509 anonymous certificates, which could then be managed used the revocation mechanisms studied in Chapter 6. In case a customer is proven to be misbehaving, then his anonymity would be revoked, preventing him from using the system. Still, the choice of the underlying group signature scheme would be critical. As we showed in Section 6.5, this choice has an unavoidable impact on the system's performance and, more importantly, in the provided functionality. Of course, the chosen group signature scheme should support both anonymity and unlinkability revocation, which is why in our tests we have used both KTY04 [132] and CPY06 [67], although other schemes could be also considered and may of course provide additional and desirable properties.

Concerning the distribution of the identities, we have stated in Section 7.2.5 that we assume customers to receive a smart card by first physically visiting the bank. This smartcard would contain their anonymous digital identities. However, note that an alternative method could be used, thanks to the protocol proposed in Chapter 5. Namely, during the physical visit of the customer to the bank[6], the bank staff could give her a random single-use activation code (associated to her bank account). This random single-use activation code would in fact be a perfectly secure implementation of the token named $ticket_{SSL}$ in Chapter 5. By means of this slight modification to the SEBIA protocol presented in Chapter 5, it could be employed for the remote distribution of the anonymous digital identities required by our online shopping system. Additionally, in this manner, the customer could actually obtain a much more flexible identification token, not requiring a smartcard reader (although keeping it secure would probably require higher security awareness from the customers).

7.6.2 Experimental results

In order to demonstrate the applicability of our approach, we show the results obtained through a prototype incorporating all the described functionality. We then compare them with current industry (working) systems. Note however that the latter systems are highly optimized ones, using distributed infrastructures. The building blocks we have used are: BFPV13 [43] for blind signatures, CPY06 [67] as group signatures (more exactly, a traceable signature scheme), and Pedersen commitments [161] and SKREP as defined in [57] for proving correctness in ZK of the commitments and the various ZK proofs.

As testing environment, we have used a laptop (Intel Core i5-480M, 4GB DDR3, with Debian Wheezy) and a desktop PC (Intel Core i7-2600, 16GB DDR3, with Debian Wheezy). We center our attention in the most frequent actions: anonymous and pseudonymous checkouts. For RSA keypairs we used a module of 1024 bits, while for ECC we used 160 bit elements. Additionally, our prototype included a MySQL database where both PS and FN keep the necessary data. We also run the following

[6]Physical presence is unavoidable at some point, if high security needs to be achieved.

fraud prevention and marketing procedures: Each time a turn was requested, PS issued a $5 discount for customer having already spent over $100; it also checked that no more than 10 purchases were made during the same day and ran the AVS test. Additionally, during checkout, PS rejected any transaction of more than $1000, and FN performed billing/shipping matching. During turn-retrieval, the customer role was played by the laptop (with 4 parallel threads), and the PS role by the desktop PC (with 8 parallel threads). During checkout, the laptop ran as a customer and merchant (each process spawning 2 threads), with the desktop PC acting as PS and FN (each process spawning 4 threads). Note that the PS throughput measures the average time intervals between the time PS receives a request and the time it has processed the response, ignoring the time taken to transmit them. Finally, at the moment the tests were performed, each machine was in a different network, with the `traceroute` command showing a route of 16 hops between them, and average roundtrip time of 44 ms. The results are summarized in Table 7.2.

Action		***AL*** (secs)	T_{PS} (reqs/sec)	***Bytes*** (sent/recv)
Turn-retrieval		$1.26(\pm 0.49)$	1.03	C: 4426/793 PS: 793/4426
Checkout	**Anon.**	$1.68(\pm 0.47)$	3.01	C: 4212/291 M: 4133/4358 PS: 1549/3908 FN: 66/1403
	Pnym.	$1.83(\pm 0.43)$	2.95	C: 5850/291 M: 6828/5996 PS: 1549/6603 FN: 66/1403

Table 7.2: Summary of results. *AL* stands for round-trip Absolute Latency: total time since the customer starts the request until she receives the response. T_{PS} stands for Throughput at the Payment System's side: total number of requests successfully answered, per second (in real time, including time spent in data transfers). *Bytes* indicates the average size in bytes of the messages sent/received during the action, for each involved entity. All results are averaged over 1000 transactions of the corresponding type.

According to the *Scalability targets* section of the Bitcoin Wiki [7], PayPal handles roughly 46 transactions per second (TPS) on the average, with peaks of 100 TPS. Also, in PayPal's website[8], it is said that PayPal has 2.6 million top customers performing 98 average purchases per year (or $\sim$ 8.08 purchases per second). For mobile devices, PayPal claims to receive 3 TPS[9]. As for Bitcoin, there is an *artificial limit* of 7 TPS [7] . Also, specialized Magento installations are claimed to achieve 550 TPS using a load-balanced web server cluster [154]. Taking into account the limitations of our experiments, it is quite notable that we achieved a rate of 1.03 TPS for turn-retrieval and a rate of roughly 3 TPS for checkout (a bit higher for anonymous ones and a bit lower for pseudonymous). While quite far from the optimized Magento setting, the performance numbers

[7] `https://en.bitcoin.it/wiki/Scalability`, accessed on January 13th, 2014.

[8] *PayPal User Statistics and Trends 2012*, `https://www.paypal.com/webapps/mpp/ent-online-attract-shoppers`. Accessed January 13th, 2014.

[9] *PayPal: Mobile commerce: your customers just can't wait*, `https://www.paypal.com/ie/webapps/mpp/sell-mobile`. Accessed January 13th, 2014.

of, both, PayPal's top and mobile customers, and also Bitcoin's rates seem both achievable for our system. Even the 46 TPS global transactions from PayPal seem reachable using an optimized and distributed setting.

As for the communication costs, we may take as reference the Bitcoin data also included in its Wiki [36], where it is said that transactions vary from 0.2 to 1 KB, averaging to roughly 0.5 KB. As seen in Table 7.2, the heavier load is taken by PS and FN. PS sends roughly $793 + 1549 = 2342$ Bytes and receives roughly $4426 + 3908 = 8334$ (anonymous checkouts) and $4426 + 6603 = 11029$ (pseudonymous checkouts) Bytes. FN sends roughly 66 Bytes and receives 1403 during checkout. Hence, PS supports approximately 10 times more communication overload than Bitcoin's peers in the best comparison. This is not bad, given the overhead of cryptography used heavily in privacy oriented communication, and further, this proportion can probably be improved by further optimization.

7.7 Chapter conclusion

In this chapter, we have presented and analyzed in detail a proposal for privacy respectful online shopping. As pointed out in recent works [72, 165], this is a field that calls for robust privacy respectful solutions. Our proposal gives customers the choice to act either pseudonymously or in a completely anonymous manner. Still, customers may benefit from marketing techniques (such as marketing promotions), but cannot use this anonymity to their advantage in order to perform illegitimate actions or fraud.

One of the most important features of the proposed system, in line with the requirements of this thesis, is that it is fully compatible with the already existing infrastructure employed in actual e-commerce. As shown in Section 7.2.1, our system is composed by the same entities and follows (almost) the same information flow than in currently deployed solutions. It also includes a rich set of additional functionality that would allow merchants to configure their own online services provided via our system, also just like actual platforms. Moreover, the core cryptographic components used for creating the anonymous identities employed in the system may be directly managed through the standard-friendly mechanisms and protocols in Chapter 5 and Chapter 6. Specifically, a prototype of the system has been tested by means of the library described in Chapter 4.

This system has been analyzed using both the formal and computational approaches, as advised in previous chapters in order to obtain a maximum level of security. Moreover, since producing practical systems is a main concern in this work, the efficiency and costs of the core functionality have been measured through an experimental prototype, concluding that it bears reasonable additional costs. Furthermore, while the additional costs have an impact on usability (and seem to be reasonable), other aspects need to be taken into account for this subject. In Section 7.2.4, we have outlined how a typical transaction using our system would resemble those of current online shopping systems. In addition, in Section 7.6 we have discussed about how to make the distribution of the required credentials more intuitive. Namely, through applying a variation of our SEBIA protocol (Chapter 5). Although a final proof would require evaluations by end users, these observations seem to point that a usable implementation is possible.

Concerning possible future work, note that the online shopping system proposed in this chapter stands out for the combination it provides of privacy for customers and flexibility for merchants. In fact, it supports many of the functionality available in current online shopping systems, while greatly enhancing its privacy. Still, the ex-

periments performed to measure its feasibility cover only the core of the system. Of special interest are the extensions for merchant-issued marketing promotions and alternative payment methods. Actually implementing these additional functionality and analyzing their impact (with respect to the additional computational and communication costs) would provide definitive proof of their applicability. Also, more detailed analysis, including measurements of other properties like power consumption would also help in further optimizing the system. Finally, and concerning the obtained results, note that while we assume customer originated communications are routed through Tor (or some equivalent anonymizing network), this has not been measured. Taking this into account would help produce more realistic estimations.

In addition, we emphasize that despite having used this turn-retrieval and checkout framework in the context of e-commerce, it may be suitable to other scenarios requiring anonymity, but where some kind of traceability or additional information associated to system users is necessary. An almost straightforward application is to e-voting. In this setting, the turn-retrieval phase could be used (probably among many other tasks) to see if the voter is actually eligible for voting (checking her age, whether she belongs to the district she is trying to vote in, etc.); after succeeding in the turn-retrieval phase, the voter could then make use of her turn for running checkout, which in this case would mean actually casting the vote. Nevertheless, an e-voting setting certainly implies many subtle issues that would need to be carefully studied.

CHAPTER 8

Fair anonymity for the Tor network

Chapter based on and supported by references [81].

Tor anonymizes communications by avoiding origin and recipient to be linked. Moreover, even the recipient cannot learn the IP address of the originator by analyzing the received packets. This is achieved by re-routing the data through several intermediaries, the Onion Routers, and adding an extra layer of encryption with each one. Nevertheless, this also reduces the protection available for the addressee, since it cannot *denounce* the originator in case of misbehavior. This is certainly a factor hindering any wide acceptance of anonymizing networks (cf. the discussion in the introduction of Chapter 6). Moreover, it causes users acting legitimately to be affected by the illegitimate actions of others. For instance, in some situations legitimate users cannot access a site through Tor because that site directly bans Tor-originated traffic. This risk has already been identified by the Tor Project and, consequently, solutions are being explored for increasing the trust in traffic coming through Tor [1].

In this chapter we show how group and blind signatures could be used to extend the functionality of Tor's entry and exit nodes in order to enable the tracing and blocking of misbehaving users. This being the case, we design an access control mechanism for Tor which does not deteriorate the normal use of the Tor network by users acting legitimately, nor their privacy. Complemented with the mechanisms in Chapter 6, this privacy respectful access control mechanism could allow, for instance, blocking a user having exceeded the number of maximum connections per second (i.e., being responsible for a denial of service attempt, spam, etc.). Similarly, through the definition of suitable policies (as those studied also in Chapter 6), more general illegitimate uses could be defined when necessary and according to the needs of each specific application accessible through Tor.

As a consequence of this *fairness* mechanism, service providers would probably increase their trust in Tor, since illegitimate actions coming from Tor would presumably be reduced. For constructing such extension in an practical and flexible manner, we make its core compatible with the (anonymous) identity distribution an management mechanisms in Chapter 5 and Chapter 6.

[1] `https://blog.torproject.org/blog/call-arms-helping-internet-services-accept-anonymous-users`. Last access on March 31st, 2015.

8.1 Related work

Recall from the fair anonymity systems paragraph at Section 6.1 that there are several systems providing anonymity revocation capabilities to Tor-like systems. However, as it was therein stated, those systems usually require complex and completely new infrastructures in order to function properly, they are usually suited for specific types of misbehavior. Also, they introduce many times prohibitive costs which, in order to reduce them, would require a complete modification of the underlying cryptographic primitives. But again, given that their infrastructure is tightly associated to this cryptographic building blocks, it would require a big effort to adapt it. On the other hand, our preliminary proposal just seems to require minimal additions to the negotiation process with entry and exit nodes in the shape of added group signatures. Thus, with the extensions proposed in Chapter 6 to the X.509 infrastructure, it would be easily deployable. And more importantly, it would inherit the flexibility derived from making the cryptographic functionality independent of the technological details and vice versa. Consequently, different group signature schemes with distinct functionalities and efficiency features could be seamlessly incorporated when required.

8.2 Incorporating fairness into Tor

In order to endow Tor with fairness capabilities, the entry and exit nodes take a central role, since they are the only nodes who learn the IP addresses of the user entering the network and that of the final destination, respectively. Hence, their knowledge would be necessary to determine whether the IP trying to access the network has already been blocked, or to demonstrate that a given origin IP has accessed certain destination IP. However, when proposing modifications of those nodes we must avoid enabling attacks based on establishing a connection between them. For that purpose, we take advantage of both the way the user negotiates keys with the Tor nodes, and the properties of group and blind signatures.

Hereafter, we assume that a group has already been set up, and that there is a suitable policy established for fairly managing revocation (see Section 8.3). Similarly, we assume that the blind signature scheme has also been set up. Table 8.1 summarizes the notation used throughout the rest paper, along with some notation inherited from the description of the Tor network [91] and the notation defined for group and blind signatures and the additional cryptographic primitives in Section 2.4.

PK_{OR_i}	Public key of Onion Router i.
$enc(\cdot)_{Tor}$	Layered encryptions following the Tor protocol.
$H(\cdot)$	Application of a cryptographic hash function.
$g^{x_\cdot}$	The user's Diffie-Hellman share.
$g^{y_\cdot}$	The Diffie-Hellman share corresponding to a Tor node.
hs_K	A transcription of the handshake for key K.
$A\|B$	A concatenated with B.
σ_1	Group signature of g^{x_1} sent to entry node.
σ_2	Group signature of g^{x_2} sent to exit node.
β	Blinded version of σ_2
$\tilde{\beta}$	Blindly signed version of β
σ_3	Blind signature of σ_2

Table 8.1: Notation summary.

Our approach works by introducing variations in the way a user negotiates the

symmetric keys with the entry and exit nodes. In short, we will require the user to group-sign the message sent during negotiation with the entry and exit nodes. In addition, in order to prevent the user to employ one identity for negotiating with the entry node, and a different one with the exit node (see Section 8.3), the entry node has to blindly sign the message that the user will send to the exit node. The resulting modified handshake schemes (see [91, p. 6]) are shown below, where $\mathtt{U}_i$ denotes any arbitrary user, EN denotes the entry node and EX the exit node. During the handshake with EN, $\mathtt{U}_i$ first group-signs g^{x_1} and g^{x_2}, sends g^{x_1} to EN and also requests EN to blindly sign a group signature of g^{x_2}. If all the operations succeed, ENaccepts the connection.

Entry Node Handshake:
$\mathtt{U}_i$: $\sigma_1 \leftarrow \mathtt{GS.Sign}(g^{x_1}, mk_i)$
$\mathtt{U}_i$: $\sigma_2 \leftarrow \mathtt{GS.Sign}(g^{x_2}, mk_i)$
$\mathtt{U}_i$: $\mathtt{com} \leftarrow \mathsf{Com}(\sigma_2, r_1)$
$\mathtt{U}_i$: $(\beta, \pi) \leftarrow \mathtt{BGS.Blind}(\mathtt{com}, r_2)$
$\mathtt{U}_i$: $\phi \leftarrow \mathtt{ShowZK}(x, w)$ where
$x = (\beta, \pi, \sigma_1), w = (mk_i, r_1, r_2)$ such that:
$\sigma_2 \leftarrow \mathtt{GS.Sign}(g^{x_2}, mk_i)$,
$(\beta, \pi) \leftarrow \mathtt{BGS.Blind}(\mathsf{Com}(\sigma_2, r_1), r_2)$
$\mathtt{U}_i \rightarrow$ EN: $g^{x_1}, \sigma_1, \beta, \pi, \phi$
EN: $\mathtt{VerifyZK}(\beta, \pi, \phi, \sigma_1)$
EN: $\mathtt{GS.Verify}(\sigma_1, g^{x_1})$
EN: $\tilde{\beta} \leftarrow \mathtt{BGS.Sign}(\beta, sbk)$
EN: $K_1 = g^{x_1 y_1}$
EN$\leftarrow \mathtt{U}_i$: $g^{y_1}, \tilde{\beta}, H(K_1 | hs_{K_1})$
$\mathtt{U}_i$: $\sigma_3 \leftarrow \mathtt{BGS.Unblind}(\tilde{\beta}, r_2)$
$\mathtt{U}_i$: $K_1 = g^{x_1 y_1}$

When $\mathtt{U}_i$ initiates the handshake with EX, she sends the group signature on g^{x_2} that was blindly signed by EN, along with the blind signature itself. If all the verifications succeed, then EX accepts the connection.

Exit Node Handshake:
$\mathtt{U}_i \rightarrow$ EX: $g^{x_2}, \sigma_2, \sigma_3$
EX: $\mathtt{GS.Verify}(\sigma_2, g^{x_2})$
EX: $\mathtt{BGS.Verify}(\sigma_3, \sigma_2)$
EX: $K_2 = g^{x_2 y_2}$
EX$\rightarrow \mathtt{U}_i$: $g^{y_2}, H(K_2 | hs_{K_2})$
$\mathtt{U}_i$: $K_2 = g^{x_2 y_2}$

It is important to note that the group signatures are encrypted using the public keys of either the entry or exit nodes. Hence, only the entry and exit nodes learn them. Moreover, the group signature sent to the exit node is blindly signed by the entry node. Thus, even if both nodes collude, they would not be able to determine by themselves that the group signatures they have received have been issued by the same user, due to the unlinkability property of the group signature scheme and the blindness property of the blind signature scheme. Moreover, since the group signature sent to the exit node has been blindly signed by the entry node, it is not possible for a user $\mathtt{U}_i$ to frame another user $\mathtt{U}_j$.

The modified key negotiation with the entry node is depicted in Fig. 8.1, and the one corresponding to the exit node is depicted in Fig. 8.2.

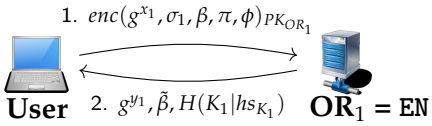


Figure 8.1: The user sends to the entry node a group signature of her share of the key, encrypted with the node's public key, and a blinded version of the group signature to be sent to the exit node. The entry node returns a blindly signed version of the latter.

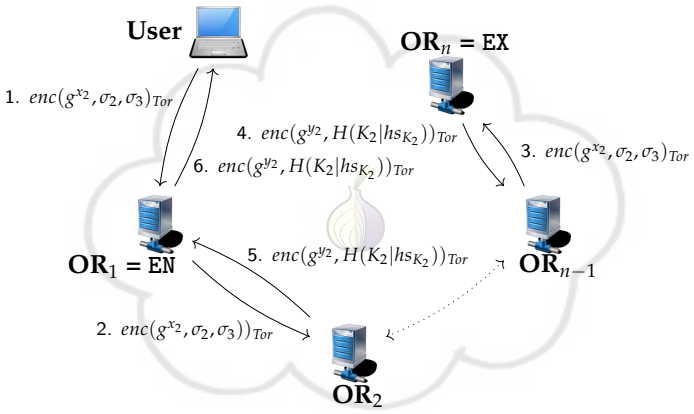


Figure 8.2: The user sends to the exit node a group signature of her share of the key, encrypted with the node's public key and the blind signature issued by the entry node.

8.2.1 How to block misbehaving users

Let us assume that some user $\mathtt{U}_i$ has been revoked due to some illegitimate behavior. When $\mathtt{U}_i$ tries to establish a circuit, he/she will need to perform a handshake with the chosen Tor entry node. Hence, upon receiving the first message with the group signature, the entry node will verify the received group signature, checking whether or not the member who issued it has been revoked. Given that the member key of $\mathtt{U}_i$ has been revoked, the verification will fail, and the entry node will reject the connection. Note that if the user has not been revoked, the privacy guarantees provided by Tor are not diminished.

8.2.2 How to denounce misbehaving users

In this case, we assume that $\mathtt{U}_i$ has already established a circuit and she is communicating with some server S (external to Tor). Also, let us suppose that eventually, $\mathtt{U}_i$ performs some illegitimate action. When that happens, S denounces this behavior following some predefined method. If deemed appropriate, the group signature received by the exit node during the handshake may be used to retrieve $\mathtt{U}_i$'s identity, or to trace her. Specifically, the exit node provides the following information:

- $enc(msg)_K$, where msg is the message received and denounced by S, and K is the symmetric key negotiated between $\mathtt{U}_i$ and the exit node.

- $(K = g^{x_2 y_2}, g^{x_2})$, where g^{x_2} is $\mathtt{U}_i$'s share of the handshake and g^{y_2} is the share created by the exit node.
- σ_2, i.e., a group signature of g^{x_2} issued by $\mathtt{U}_i$.

In order to verify that the received denounce is valid, it is necessary to check that the message received from S, msg, corresponds to the encryption $\{msg\}_K$ received from the exit node. Also, σ_2 must be a valid group signature over g^{x_2}. Finally, the exit node may be required to prove that it knows the discrete logarithm y_2 of $g^{x_2 y2}$ to the base g^{x_2}. If these checks succeed, then the member with key mk_i ($\mathtt{U}_i$) is responsible of msg. Hence, $\mathtt{U}_i$'s key can be consequently revoked, and the circuit may be closed by the exit node. Note that subsequent attempts made by $\mathtt{U}_i$ to establish a circuit would be blocked by the entry node, since the member key of $\mathtt{U}_i$ has been revoked.

8.2.3 Additional remarks

A few remarks are worth to be made, concerning "special" situations and subjects that should be taken into account.

Tor bridges Our proposal is directly extensible to support Tor bridges, considering Tor bridges as the entry nodes to the Tor network.

Leaky pipe topology This approach allows a client to output some packets through a node other than the negotiated exit. In order to adapt our approach to this exception, the node leaking the packet should follow ad-hoc the procedure defined for exit nodes.

Logging Since hosts being accessed through these *fair* Tor extension might want to issue denounces against traffic originated from Tor, logging the information necessary for solving disputes is required. Hence, the exit nodes need to keep the information specified in Section 8.2.2 (this also applies to the leaky pipe extension). An appropriate policy for logging should be established.

Denunciation time span Considering that our proposal requires Tor exit nodes (and circumstantially other nodes) to log several pieces of information, it would comprise a serious scalability problem if this logging would be expected to last too much time. Hence, it seems appropriate to establish a predetermined time span for accepting denounces. Upon expiration of that time span, all the logged information could be removed, and any subsequent denounce related to that information rejected. This would require possible complainants to be aware of this time limitation.

8.3 Open issues

In the scheme given in Section 8.2 we just use the general definitions of the building blocks for defining our system. The analysis of which specific variants should be employed is left as future work. Note that this is a very delicate decision, since different options offer different privacy properties. Moreover, we may even need different schemes depending on who issues the signatures (e.g. group signatures are issued both by users and entry points in our proposal). Thus, given the sensitivity of the information managed by Tor, this is an issue that needs to be studied in depth by itself. For

that matter, the extensible group signatures library `libgroupsig` presented in Chapter 4 may offer interesting features. In addition, concerning the blind signatures, it would probably be necessary to use some of its variant to prevent circumventing the controls explaining above. Namely, with the previous bare scheme, a user could use the same blind signature indefinitely. This may simply be solved by using partially blind signatures, and having the entry node introduce a *lifetime* value for the blind signature as common message.

Another important issues are determining when misbehaving users should be revoked, and by whom. The former question would probably depend on the websites (or service) being accessed through Tor. For the latter, a probably good solution given Tor's infrastructure would be to apply threshold schemes to the revocation procedures (see [32]), such that a majority of the authorities participating in the network consensus need to agree for revoking users.

Finally, note that Sybil attacks [94] are partly addressed by forcing users to use the same member key for the group signature sent to the entry node and for the group signature sent to the exit node (and having the latter to be blindly signed by the entry node). However, some additional mechanism should be included for preventing users from arbitrarily generating new member keys. Since asking users to register may not be well received (it may be seem as a threat to anonymity), requesting them to perform some proof of work [95] during the generation of the member keys may be a good alternative.

8.3.1 Further work

While this extension would help in reducing the mistrust that service providers might have in the Tor network and the traffic coming from it, there are several aspects that need to be very carefully studied.

First, the main objective of Tor is to ensure privacy (through anonymity). Thus, adding revocation capabilities would probably not be well received, even if it is aimed to provide a better service to legitimate users. In this subject, a detailed analysis for choosing the most suitable group and blind signature schemes for ensuring that the revocation functionality cannot be misused too. Specifically, techniques such as objective blacklisting [112] and trust distribution [32] would certainly reduce the controversy. Additionally, the task for creating and distributing the member keys employed for issuing the group signatures is a critical task. The protocol in Chapter 5 can be used for this, but defining when and how to create new identities is necessary for maintaining an adequate balance between anonymity and costs.

Second, a formal analysis of the trust and attack models, and the main use cases (e.g. stating the basic misbehaviors) would help locating additional required functionality and provide further security guarantees.

Also, although the same principle for misuse detection might be applied for Tor Hidden Services[2], in depth analysis of the use case scenario is necessary in order to add support for these services. This would actually be an important addition since, as we discussed in the introduction of Chapter 6, Tor Hidden Services are many times used for hosting illegitimate activities.

Finally, once the previous open issues are satisfactorily addressed, experimental measurements of the additional costs introduced by our proposal would be necessary

[2] `https://www.torproject.org/docs/hidden-services.html.en`. Last access on March 31st, 2015.

in order to meet the efficiency requirements of a low-latency anonymity network such as Tor.

8.4 Chapter conclusion

The extension to the Tor network proposed in this chapter would endow it with the functionality for preventing misbehaving users to access the network. We expect such functionality to increase the trust of websites in Tor and thus prevent them to block users coming from it.

This extension follows the design of Tor, and does not require any modification to its infrastructure. It works by including group signatures in the key negotiation processes with the entry and exit nodes and having the entry node blindly sign the group signature to be sent to the exit node. The group signature sent to the exit node allows service providers to denounce illegitimate actions without learning their identity. Once the unlinkability of a user has been revoked as a consequence of some illegitimate behavior, any entry node would be able to block that specific user just by checking if it is included in an (unlinkability) Revocation List.

While the specific details on how to create and distribute the member keys required for the group signatures has been left as future work, this task can certainly be solved using the protocol in Chapter 5. Also, the mechanisms in Chapter 6 compose the necessary framework for managing anonymity in a fair manner. Finally, the implementation of this extension would certainly be eased through the library in Chapter 4.

Conclusion and summary of contributions

Throughout this work, we have followed an incremental approach for addressing our main goal of easing the deployment of privacy respectful systems through anonymity. Specifically, we have first proposed a set of basic elements and protocols that jointly compose a framework suitable for achieving a reasonably high level of privacy and security within current technological infrastructures. Moreover, we have given examples of specific systems that may be built with the help of this framework. While further work is certainly necessary in order to refine the additional costs introduced by our proposals and in order to keep extending the privacy respectful functionality achievable through them, we have also included empirical results showing its feasibility by means of prototypes developed by ourselves.

Next, we describe in finer detail each of the main contributions of this thesis.

Summary of results

Recalling the main objectives of this work, our major goal was to help easing the task of designing and implementing privacy respectful but deployable (and thus practical) systems. Specifically, this has been addressed as follows. In Part II of this thesis we have tackled the first two objectives as stated in Section 1.1:

1. **Design methodology**. The methodology in Chapter 3 does help in the design of secure systems and in their subsequent verification. Specifically, it is aimed to help in the "translation" of informal requirements into formal ones. We have employed it in most of the protocols and systems introduced in this thesis.

2. **Implementing privacy through anonymity**. Through the contribution of the C library for group signatures in Chapter 4, we have overcome one main difficulty for actually implementing all our subsequent proposals, which address privacy through anonymity. Moreover, since it is an open source and extensible library, providing a unified API, we expect it to be used by the community as a building block for this kind of systems.

In Part III and Part IV, we have addressed the remaining objectives, which are actually based in the previous results. We have proposed protocols and mechanisms dealing with anonymous identities management in fair anonymity systems, and further designed privacy respectful systems compatible with them in the context of e-commerce and Tor, the anonymous communications network. Specifically, our results have been:

3. **Secure and usable distribution of identities**. This is directly achieved throughout SEBIA, the protocol in Chapter 5. Moreover, it follows the design principles of EBIA, the most widely deployed online registration protocol. Additionally, it is implementable with current technologies and within the existing infrastructures.

4. **Mechanisms for managing privacy enabling identities**. This functionality is directly provided by the mechanisms in Chapter 6, which allow anonymous identities to match the managing options of conventional identities. Since they are based on the X.509 infrastructure, the costs for actually deploying them in existing systems would be kept to a reasonable minimum.

5. Also, in Part IV, we have proposed two systems that are fully compatible with the previous proposals. More concretely:

   (a) **Caduceus**. A privacy respectful online shopping framework following the infrastructure and information flow of current industry systems. Through anonymity, achieved using group and blind signatures, the system is compatible with the typical marketing and fraud prevention techniques. We implement a prototype of this system with the help of `libgroupsig` (Chapter 4). Both the anonymous identity distribution and management procedures could be implemented in practice with the protocols and mechanisms in Chapter 5 and Chapter 6, respectively. Finally, the security of this system is verified using the methodology in Chapter 3, both with the formal and computational approach.

   (b) **Fairness extensions to Tor**. Also based in group and blind signatures, we propose an extension to the handshake procedures with entry and exit nodes in the Tor network that allow to block misbehaving users in a privacy respectful manner. The core parts of this system would thus be implementable with the library in Chapter 4, and the distribution and management of the anonymous identities through the proposals in Chapter 5 and Chapter 6. The verification of its security properties (with Chapter 3) is left as future work.

Thus, summarizing, in Chapter 3 and Chapter 4 we have first set the grounds for designing and implementing privacy respectful and secure systems through a secure design methodology and a library for group signatures, which we use as a core for providing anonymity; subsequently, we have proposed in Chapter 5 and Chapter 6 several protocols for addressing the most common tasks; and finally, in Chapter 7 and Chapter 8, we have described two systems that may be built based on our previous proposals. These works may act as fundmental building blocks for privacy respectful e-democracy applications such as e-voting, online surveys, open data, but also providing fairness (accountability) when necessary. Moreover, the technology upon which our protocols and systems (or any application derived from them) may be implemented, outlined in Table 8.2, is already in wide use, which actually eases our proposals' deployment.

Future work

We have outlined specific further work within each of the chapters of this thesis. However, some concrete lines stand out, given the main goal of this work. In more detail,

Proposed protocol/system	Technology for implementation	Description	Chapter
SEBIA	EBIA	Secure email-based registration protocol	Chapter 5
Anonymity management	X.509	CRL extension OCSP extension ACFP protocol	Chapter 6
Caduceus	SEBIA X.509	Privacy respectful online shopping	Chapter 7
FairTor	SEBIA X.509	Fair anonymous access control in Tor	Chapter 8

Table 8.2: Technology upon which the proposed protocols and systems may be built.

there is a lack of support (through actual implementations) of advanced privacy enhancing cryptographic primitives, as we have pointed out in [23]. While we have provided a library for group signatures in Chapter 4, implementations for the primitives reviewed in [23], following the same design principles as `libgroupsig` would certainly constitute a good contribution to the computer science and cryptographic communities. Specially, taking into consideration the challenges summarized in [21].

Regarding the extensions to X.509 in Chapter 6, the next step should be to submit the proposal to a standardization body, which would certainly help in gaining insight into possible additional extensions or refinements. Subsequently, actually implementing X.509 anonymous identities (through, for instance, `ligroupsig`) and our proposals for extending the X.509 management capabilities would undoubtedly suppose a big contribution towards our main goal of easing the deployment of privacy respectful systems.

Finally, a major line of future work is related to the work in Chapter 7. It is our opinion that the design of the system therein described is of special interest by itself and worth further study for finding useful applications into contexts other than e-commerce. Specifically, the separation of the overall transaction in two unlinkable but interdependent pseudonymous and anonymous phases may prove to be a useful construction[3] when it is necessary to perform sensitive actions that require both user profiling and user privacy. For instance, in contexts such as e-voting, social networks, surveys or cloud services. Furthermore, its derivation towards the scenario studied in Chapter 8, where the initial pseudonymous phase is substituted by another anonymous phase[4] also seems promising when pseudonymity is not acceptable, yet access control mechanisms are advisable. In this latter case, while both phases are anonymous, different verifications are necessary in each of them, what still makes unlinkability a requirement.

Altogether, providing further support in the shape of source code for more advanced cryptographic primitives and our X.509 extensions, as well as for our SEBIA protocol, would certainly contribute towards the development of privacy respectful systems. Specifically, through facilitating the creation and distribution of anonymous digital identities and their subsequent management. Additionally, applying them to systems adopting the pseudonymous-anonymous (or anonymous-anonymous) constructions would probably constitute a powerful and flexible combination. And last,

[3] We may refer to it as a pseudonymous-anonymous two-phase transaction.
[4] This special case could be referred to as an anonymous-anonymous two-phase transaction.

using our security verification methodology (Chapter 3) in order to ensure the security of the resulting proposals would, on one hand, raise the achieved security level and, on the other hand, help in the refinement and further development of our methodology.

Contributions

The following publications have been produced during this thesis, and thus support its contents:

Publications in peer-reviewed conferences and journals.

- David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Non-conventional digital signatures and their implementations – a review. Accepted in CISIS, 2015. CORE B (2013).
- David Arroyo, Jesus Diaz, and Victor Gayoso. On the difficult tradeoff between security and privacy: challenges for the management of digital identities. Accepted in CISIS, 2015. CORE B (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. New X.509-based mechanisms for fair anonymity management. Computers & Security, 46:111–125, 2014. Impact Factor 1.172, JCR Q2 (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Methodological security verification of a registration protocol. In CISIS, pages 214–221, 2014. CORE B (2013). Acceptance Rate 35%.
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. On securing online registration protocols: Formal verification of a new proposal. Knowl.-Based Syst., 59:149–158, 2014. Impact Factor 3.058, JCR Q1 (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. A formal methodology for integral security design and verification of network protocols. Journal of Systems and Software, 89:87–98, 2014. Impact Factor 1.245, JCR Q2 (2013).
- David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Cryptanalysis of a one round chaos-based substitution permutation network. Signal Processing, 93(5):1358–1364, 2013. Impact Factor 2.238, JCR Q1 (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Anonymity revocation through standard infrastructures. In EuroPKI, pages 112–127, 2012. CORE B (2013). Acceptance Rate 40%.
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. An approach for adapting Moodle into a secure infrastructure. In CISIS, pages 214–221, 2011. CORE B (2013). Acceptance Rate 54%.

Publications currently submitted to peer-reviewed journals.

- Jesus Diaz, David Arroyo, Francisco B. Rodriguez. `libgroupsig`: An extensible C library for group signatures, 2014.

Publications in non peer-reviewed public repositories.

- Jesus Diaz, David Arroyo, Francisco B. Rodriguez. Fair anonymity for the Tor network. Available at `http://arxiv.org/abs/1412.4707`, 2014.

Conclusión

En este trabajo, hemos seguido una aproximación incremental para alcanzar nuestro objetivo de facilitar el despliegue de sistemas respetuosos con la privacidad, a través del anonimato. Específicamente, hemos propuesto primero un conjunto de elementos básicos y de protocolos que, conjuntamente, permiten crear un marco de trabajo apropiado para conseguir unos niveles de privacidad y seguridad razonablemente altos, utilizando infraestructuras tecnológicas actuales. Más aun, hemos dado ejemplos de sistemas específicos que pueden ser diseñados e implementados gracias a este marco de trabajo. Aunque es necesario trabajo adicional para refinar los sobrecostes introducidos por nuestras propuestas y con el fin de seguir ampliando las funcionalidades respetuosas con la privacidad que proporcionan, hemos incluido también medidas empíricas, obtenidas a través de prototipos implementados por nosotros mismos, mostrando que los costes adicionales son razonables.

A continuación, describimos en más detalle cada una de las contribuciones de esta tesis.

Resumen de resultados

Recordando los principales objetivos de la tesis, la principal meta era facilitar las tareas de diseño y desarrollo de sistemas respetuosos con la privacidad, que al mismo tiempo fueran fácilmente desplegables (y por lo tanto, prácticos). Específicamente, para este fin, en la Parte II de esta tesis hemos tratado los dos objetivos iniciales establecidos en la Sección 1.1.

1. **Metodología de diseño**. La metodología en Capítulo 3 ayuda durante el diseño de sistemas seguros y su posterior verificación. En concreto, está destinada a ayudar en la "traducción" de requisitos informales en requisitos formales. Hemos utilizado esta metodología en muchos de los protocolos y sistemas introducidos en esta tesis.

2. **Implementando privacidad a través de anonimato**. Con la contribución de una librería de firmas grupales, en el Capítulo 4, hemos solucionado una de las principales dificultades para la implementación de nuestras propuestas en los capítulos posteriores, que proporcionan privacidad a través de anonimato. Esta librería se ha publicado como código abierto, es fácilmente extensible, y proporciona una API unificada, con lo que esperamos que sea de utilidad como pilar básico para la implementación de este tipo de sistemas en la comunidad criptográfica.

En la Parte III y la Parte IV, hemos tratado los objetivos restantes, que a su vez se basan en los resultados anteriores. Hemos propuesto protocolos y mecanismos para

poder distribuir y gestionar las identidades anónimas necesarias en sistemas de anonimato justo. Además, hemos diseñado sistemas respetuosos con la privacidad, compatibles con dichos protocolos y mecanismos, en el contexto de comercio electrónico y Tor, la red de comunicaciones anónimas más popular. En concreto, nuestros resultados son:

3. **Distribución segura y usable de identidades digitales**. Esto se consigue directamente a través de SEBIA, el protocolo diseñado en el Capítulo 5. Además, SEBIA sigue los principios de diseño de EBIA, el protocolo para registro online más desplegado en la actualidad. SEBIA es implementable con tecnologías actuales y utilizando infraestructuras ya existentes.

4. **Mecanismos para gestionar identidades digitales respetuosas con la privacidad**. Esta funcionalidad la proporcionan directamente los mecanismos en el Capítulo 6, que permiten alcanzar el mismo conjunto de funcionalidades disponibles para la gestión de identidades digitales convencionales, pero para el caso de identidades digitales anónimas. Además, al estar basadas en la infraestructura X.509, los costes necesarios para desplegar estos mecanismos en los sistemas actuales se mantienen en un mínimo razonable.

5. Además, en la Parte IV, hemos propuesto dos sistemas que son totalmente compatibles con las propuestas anteriores. Específicamente:

   (a) **Caduceus**. Un sistema que proporciona un marco de trabajo para desarrollar plataformas de compras online respetuosas con la privacidad, que se adapta a las infraestructuras y al flujo de información de los sistemas actuales. Utilizando el anonimato como base, conseguido gracias a firmas grupales y firmas parcialmente ciegas, el sistema es compatible con las típicas técnicas de marketing y prevención de fraude. Además, hemos implementado un prototipo de este sistema con la ayuda de `libgroupsig` (Capítulo 4). Los procedimientos tanto para la distribución como para la gestión de las identidades anónimas necesarias podrían implementarse con los protocolos y mecanismos en el Capítulo 5 y el Capítulo 6, respectivamente. Por último, la seguridad de este sistema se ha comprobado utilizando la metodología en el Capítulo 3, tanto en el modelo formal como en el modelo computacional.

   (b) **Extensiones de anonimato justo para Tor**. Basándonos también en firmas grupales y firmas ciegas, proponemos una extensión sobre los mecanismos de negociación de claves con los que los nodos de entrada y salida de la red Tor podrían bloquear a usuarios que actuasen deshonestamente, pero respetando la privacidad del sistema. Los componentes principales de este sistema se pueden implementar con la librería en el Capítulo 4, y la distribución y gestión de identidades anónimas a través de las propuestas en el Capítulo 5 y el Capítulo 6. La verificación de sus propiedades de seguridad (utilizando la metodología en el Capítulo 3) se deja como trabajo futuro.

Resumiendo, en el Capítulo 3 y el Capítulo 4 hemos establecido las bases para el diseño y la implementación de sistemas seguros y respetuosos con la privacidad, mediante una metodología para diseño seguro y una librería de firmas grupales, que utilizamos como componente principal para proporcionar anonimato; a continuación, hemos propuesto en el Capítulo 5 y el Capítulo 6 varios protocolos y mecanismos que permiten resolver, de manera respetuosa con la privacidad, las tareas más comunes en

sistemas basados en Internet; finalmente, en el Capítulo 7 y el Capítulo 8, hemos descrito dos sistemas que se pueden construir a partir de nuestras propuestas anteriores. Además, las tecnologías sobre la que se basan nuestros protocolos y sistemas, resumida en la Tabla 8.2, ya se encuentra ampliamente extendida, lo cual facilitaría el despliegue de nuestras propuestas.

Protocolo/sistema propuestos	Tecnología para su implementación	Descripción	Capítulo
SEBIA	EBIA	Protocolo para registro seguro basado en emails	Chapter 5
Gestión de anonimato	X.509	Extensión a CRL Extensión a OCSP Protocolo ACFP	Chapter 6
Caduceus	SEBIA X.509	Compras online respetuosas con la privacidad	Chapter 7
FairTor	SEBIA X.509	Control de acceso respetuoso con la privacidad en Tor	Chapter 8

Table 8.3: Tecnología en la que se basan nuestros protocolos y sistemas.

Trabajo futuro

En cada capítulo de esta tesis hemos indicado posibles líneas de trabajo futuro específicas. No obstante, algunas de ellas son especialmente relevantes, teniendo en cuenta el objetivo principal de esta tesis. Con algo más de detalle, se ha visto que hay una falta de soporte (en forma de implementación de sistemas) para primitivas criptográficas avanzadas destinadas a mejorar la privacidad, como hemos indicado en [23]. En esta tesis hemos introducido una librería de firmas grupales en el Capítulo 4. Siguiendo los mismos principios de diseño que dicha librería, proporcionar implementaciones de las primitivas revisadas en [23] supondría sin lugar a dudas una importante contribución a las comunidades criptográficas e informáticas. Especialmente, teniendo en cuenta los desafíos enumerados en [21].

En cuanto a las extensiones a X.509 en el Capítulo 6, el siguiente paso sería enviar la propuesta a algún organismo de estandarización, lo cual ayudaría sin duda a obtener mayor *feedback* acerca de posibles mejoras o extensiones. A continuación, la implementación de identidades anónimas basadas en X.509 (a través de `libgroupsig`, por ejemplo) y de nuestras propuestas para la gestión de las mismas, supondría una importante contribución hacia nuestro objetivo de facilitar la implantación de sistemas respetuosos con la privacidad.

Por último, una importante línea de trabajo futuro se centraría en el trabajo descrito en el Capítulo 7. En nuestra opinión, el diseño del sistema propuesto es de especial interés. En concreto, la separación de la transacción global del sistema en dos fases (una pseudónima y otra anónima), no enlazables entre sí pero interdependientes, probablemente sea una arquitectura[5] útil para contextos en los que es necesario realizar acciones sensibles que requieran al mismo tiempo incorporar controles de los perfiles de los usuarios y medidas de privacidad. Por ejemplo, en sistemas de votación electrónica, redes sociales, encuestas o sistemas basados en la nube. Más aún, su derivación hacia el

[5]Podemos referirnos a ella como una transacción pseudónima-anónima en dos fases.

escenario estudiado en el Capítulo 8, donde la fase pseudónima inicial se sustituye por otra fase anónima[6] también parece interesante para los casos en los que la pseudonimia no sea aceptable. En concreto, en este caso, aunque las dos fases son anónimas, cada una de ellas requiere distintas verificaciones, siendo también necesario que no sean enlazables entre sí.

En conjunto, proporcionando mayor soporte en forma de código fuente para primitivas criptográficas avanzadas, y nuestras extensiones a X.509, así como el protocolo SEBIA, sin lugar a dudas contribuiríamos al desarrollo de sistemas respetuosos con la privacidad. En concreto, facilitando la creación y distribución de identidades digitales anónimas y su posterior gestión. Además, aplicando estas propuestas sobre sistemas con arquitecturas de tipo transacción en dos fases pseudónima-anónima (o anónima-anónima) supondría una combinación potente y flexible. Por último, utilizando nuestra metodología de verificación (Chapter 3) con el fin de alcanzar mayores garantías de seguridad ayudaría también a refinar y seguir desarrollando la propia metodología.

Contribuciones

Las siguientes publicaciones han sido producidas durante esta tesis, soportando por lo tanto sus contenidos:

Publicaciones en conferencias y revistas revisadas por pares.

- David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Non-conventional digital signatures and their implementations – a review. Accepted in CISIS, 2015. CORE B (2013).
- David Arroyo, Jesus Diaz, and Victor Gayoso. On the difficult tradeoff between security and privacy: challenges for the management of digital identities. Accepted in CISIS, 2015. CORE B (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. New X.509-based mechanisms for fair anonymity management. Computers & Security, 46:111–125, 2014. Impact Factor 1.172, JCR Q2 (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Methodological security verification of a registration protocol. In CISIS, pages 214–221, 2014. CORE B (2013). Acceptance Rate 35%.
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. On securing online registration protocols: Formal verification of a new proposal. Knowl.-Based Syst., 59:149–158, 2014. Impact Factor 3.058, JCR Q1 (2013).
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. A formal methodology for integral security design and verification of network protocols. Journal of Systems and Software, 89:87–98, 2014. Impact Factor 1.245, JCR Q2 (2013).
- David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Cryptanalysis of a one round chaos-based substitution permutation network. Signal Processing, 93(5):1358–1364, 2013. Impact Factor 2.238, JCR Q1 (2013).

[6] Podemos refereirnos a esta construcción como una transacción anónima-anónima en dos fases.

- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Anonymity revocation through standard infrastructures. In EuroPKI, pages 112–127, 2012. CORE B (2013). Acceptance Rate 40%.
- Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. An approach for adapting Moodle into a secure infrastructure. In CISIS, pages 214–221, 2011. CORE B (2013). Acceptance Rate 54%.

Publicaciones actualmente enviadas a revistas revisadas por pares.

- Jesus Diaz, David Arroyo, Francisco B. Rodriguez. `libgroupsig`: An extensible C library for group signatures, 2014.

Publicaciones en repositorios públicos no revisados por pares.

- Jesus Diaz, David Arroyo, Francisco B. Rodriguez. Fair anonymity for the Tor network. Available at `http://arxiv.org/abs/1412.4707`, 2014.

Bibliography

[1] *Digital Privacy: Theory, Technologies, and Practices*. Auerbach Publications, 1 edition, December 2007. (Cited on page 16)

[2] Common criteria for information technology security evaluation – part 3: Security assurance components. Technical report, July 2009. (Cited on page 35)

[3] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115, 2001. (Cited on page 18)

[4] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999. (Cited on page 17)

[5] Martín Abadi and Roger M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996. (Cited on page 21, 85)

[6] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, pages 3–22, 2000. (Cited on page 16, 21)

[7] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In *ASIACRYPT*, pages 244–251, 1996. (Cited on page 25)

[8] Alessandro Acquisti. The economics of personal data and the economics of privacy, 2010. (Cited on page 33)

[9] Ben Adida. Beamauth: two-factor web authentication with a bookmark. In *ACM Conference on Computer and Communications Security*, pages 48–57, 2007. (Cited on page 71)

[10] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, pages 119–135, 2001. (Cited on page 123)

[11] Mihhail Aizatulin, François Dupressoir, Andrew D. Gordon, and Jan Jürjens. Verifying cryptographic code in c: Some experience and the csec challenge. In *Formal Aspects in Security and Trust*, pages 1–20, 2011. (Cited on page 35)

[12] Mihhail Aizatulin, Andrew D. Gordon, and Jan Jürjens. Extracting and verifying cryptographic models from c protocol code by symbolic execution. In *ACM Conference on Computer and Communications Security*, pages 331–340, 2011. (Cited on page 35)

[13] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013. (Cited on page 56)

[14] Joseph Ayo Akinyele. *Enabling Machine-aided Cryptographic Design*. PhD thesis, Johns Hopkins University, 2013. (Cited on page 35, 36)

[15] Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems (2. ed.)*. Wiley, 2008. (Cited on page 16)

[16] Ross J. Anderson and Roger M. Needham. Robustness principles for public key protocols. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, pages 236–247, 1995. (Cited on page 21)

[17] Christian Haack Alan Jeffrey Andrew D. Gordon. Cryptyc: cryptographic protocol type checker. `http://cryptyc.cs.depaul.edu/`, 2002. (Cited on page 17, 41, 42)

[18] Elli Androulaki and Steven M. Bellovin. Apod: Anonymous physical object delivery. In *Privacy Enhancing Technologies*, pages 202–215, 2009. (Cited on page 124, 135)

[19] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *Privacy Enhancing Technologies*, pages 202–218, 2008. (Cited on page 123)

[20] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *Financial Cryptography*, pages 34–51, 2013. (Cited on page 136)

[21] David Arroyo, Jesus Diaz, and Victor Gayoso. On the difficult tradeoff between security and privacy: challenges for the management of digital identities. In *CISIS*, 2015 – In press. (Cited on page 55, 93, 153, 159)

[22] David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Cryptanalysis of a one round chaos-based substitution permutation network. *Signal Processing*, 93(5):1358–1364, 2013. (Cited on page 17, 33)

[23] David Arroyo, Jesus Diaz, and Francisco B. Rodriguez. Non-conventional digital signatures and their implementations – a review. In *CISIS*, 2015 – In press. (Cited on page 55, 153, 159)

[24] Man Ho Au, Patrick P. Tsang, and Apu Kapadia. PEREA: Practical TTP-free revocation of repeatedly misbehaving anonymous users. *ACM Trans. Inf. Syst. Secur.*, 14(4):29, 2011. (Cited on page 96)

[25] Ero Balsa, Laura Brandimarte, Alessandro Acquisti, Claudia Diaz, and Seda Gurses. Spiny CACTOS: OSN Users Attitudes and Perceptions Towards Cryptographic Access Control Tools. *Proceedings 2014 Workshop on Usable Security*, 2014. (Cited on page 6)

[26] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *POPL*, pages 90–101, 2009. (Cited on page 17)

[27] Deborah L. Bayles and Hamir Bhatia. *E-Commerce Logistics & Fulfillment: Delivering the Goods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. (Cited on page 124)

[28] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT*, pages 259–274, 2000. (Cited on page 16)

[29] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 244–250, 1993. (Cited on page 17)

[30] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. *ACM Trans. Program. Lang. Syst.*, 33(2):8, 2011. (Cited on page 35)

[31] Vicente Benjumea, Seung Geol Choi, Javier Lopez, and Moti Yung. Anonymity 2.0 - x.509 extensions supporting privacy-friendly authentication. In *CANS*, pages 265–281, 2007. (Cited on page 56, 62, 95, 97, 128, 138)

[32] Vicente Benjumea, Seung Geol Choi, Javier Lopez, and Moti Yung. Fair traceable multi-group signatures. In *Financial Cryptography*, pages 231–246, 2008. (Cited on page 23, 56, 108, 109, 110, 148)

[33] John Bethencourt, Elaine Shi, and Dawn Song. Signatures of reputation. In *Financial Cryptography*, pages 400–407, 2010. (Cited on page 123)

[34] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *POPL*, pages 445–456, 2010. (Cited on page 35)

[35] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Content and popularity analysis of tor hidden services. *CoRR*, abs/1308.6768, 2013. (Cited on page 93)

[36] Bitcoin. Bitcoin scalability, 2014. (Cited on page 140)

[37] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96, 2001. (Cited on page 20)

[38] Bruno Blanchet. From secrecy to authenticity in security protocols. In *SAS*, pages 342–359, 2002. (Cited on page)

[39] Bruno Blanchet. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *FOSAD*, pages 54–87, 2013. (Cited on page 17, 18, 20, 38, 42, 49)

[40] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008. (Cited on page 196)

[41] Bruno Blanchet and David Cadé. Cryptoverif: Cryptographic protocol verifier in the computational model. `http://www.cryptoverif.ens.fr/`, 2012. (Cited on page 17)

[42] Bruno Blanchet, Ben Smyth, and Vincent Cheval. *ProVerif Automatic Cryptographic Protocol Verifier User Manual and Tutorial*. Paris, June 2014. (Cited on page 20, 75, 83, 196)

[43] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Short blind signatures. *Journal of Computer Security*, 21(5):627–661, 2013. (Cited on page 25, 138, 201)

[44] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004. (Cited on page 56)

[45] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *ACM Conference on Computer and Communications Security*, pages 168–177, 2004. (Cited on page 99)

[46] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MOBICOM*, pages 180–189, 2001. (Cited on page 49, 51, 52)

[47] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In *CRYPTO*, pages 302–318, 1993. (Cited on page 25)

[48] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. (Cited on page 26)

[49] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Security proofs for an efficient password-based key exchange. In *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 241–250, 2003. (Cited on page 17)

[50] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, pages 132–145, 2004. (Cited on page 56)

[51] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990. (Cited on page 17)

[52] Jan Camenisch, Maria Dubovitskaya, Anja Lehmann, Gregory Neven, Christian Paquin, and Franz-Stefan Preiss. Concepts and languages for privacy-preserving attribute-based authentication. In Simone Fischer-HÃijbner, Elisabeth Leeuw, and Chris Mitchell, editors, *Policies and Research in Identity Management*, volume 396 of *IFIP Advances in Information and Communication Technology*, pages 34–52. Springer Berlin Heidelberg, 2013. (Cited on page 97)

[53] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious transfer with access control. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 131–140, New York, NY, USA, 2009. ACM. (Cited on page 123)

[54] Jan Camenisch, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Christian Paquin, Kai Rannenberg, and Harald Zwingelberg. D2.1 architecture for attribute-based credential technologies - version 1. https://abc4trust.eu/index.php/pub/results/107-d21architecturev1, 2011. (Cited on page 97)

[55] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, pages 61–76, 2002. (Cited on page 63)

[56] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. An efficient fair payment system. In *ACM Conference on Computer and Communications Security*, pages 88–94, 1996. (Cited on page 123)

[57] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997. (Cited on page 138)

[58] Sébastien Canard and Aline Gouget. Divisible e-cash systems can be truly anonymous. In *EUROCRYPT*, pages 482–497, 2007. (Cited on page 123)

[59] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Applied Mathematics*, 154(2):189–201, 2006. (Cited on page 56)

[60] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001. (Cited on page 17)

[61] Sagar Chaki and Anupam Datta. Aspier: An automated framework for verifying security protocol implementations. In *CSF*, pages 172–185, 2009. (Cited on page 35)

[62] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. (Cited on page 134)

[63] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981. (Cited on page 13, 93)

[64] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1982. (Cited on page 24, 93, 123)

[65] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991. (Cited on page 22, 55, 93, 97)

[66] Xiaofeng Chen, Fangguo Zhang, Yi Mu, and Willy Susilo. Efficient provably secure restrictive partially blind signatures from bilinear pairings. In *Financial Cryptography*, pages 251–265, 2006. (Cited on page 25)

[67] Seung Geol Choi, Kunsoo Park, and Moti Yung. Short traceable signatures based on bilinear pairings. In *IWSEC*, pages 88–103, 2006. (Cited on page 23, 24, 56, 111, 113, 138)

[68] Federal Trade Commission. Privacy online: A report to congress, 1998. (Cited on page 121)

[69] Scott E. Coull, Matthew Green, and Susan Hohenberger. Access controls for oblivious and anonymous systems. *ACM Trans. Inf. Syst. Secur.*, 14:10:1–10:28, June 2011. (Cited on page 123)

[70] D. Crocker, T. Hansen, and M. Kucherawy. Domainkeys identified mail (DKIM) signatures. RFC 6376 (Draft Standard), September 2011. (Cited on page 74, 88)

[71] George I. Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. Anonymity control in e-cash systems. In *Financial Cryptography*, pages 1–16, 1997. (Cited on page 123)

[72] Yves-Alexandre de Montjoye, Laura Radaelli, Vivek Kumar Singh, and Alex "Sandy" Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science*, The end of privacy:536–539, 2015. (Cited on page 5, 121, 123, 140)

[73] Eva M. Blanco Delgado. Diseño y desarrollo de una aplicación Android para el uso de identidades digitales, autenticación y firmas digitales en sistemas interactivos. B.S. Thesis, Escuela Politécnica Superior, UAM, Spain, 2014. (Cited on page 90)

[74] Università di Genova, CASSIS group, Information Security Group (ETHZ), and Siemens AG. Avispa: Automated validation of internet security protocols and applications, 2006. (Cited on page 17)

[75] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Privacy Enhancing Technologies*, pages 54–68, 2002. (Cited on page 16)

[76] Claudia Díaz, Omer Tene, and Seda GÃijrses. Hero or Villain: The Data Controller in Privacy Law and Technologies. *Ohio State Law Journal*, 74:im Erscheinen, 2013. (Cited on page 102, 107)

[77] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. An approach for adapting Moodle into a secure infrastructure. In *CISIS*, pages 214–221, 2011. (Cited on page 69, 89)

[78] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Anonymity revocation through standard infrastructures. In *EuroPKI*, pages 112–127, 2012. (Cited on page 60, 62, 93, 128)

[79] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. A formal methodology for integral security design and verification of network protocols. *CoRR*, abs/1201.5666, 2012. (Cited on page 37)

[80] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Formal security analysis of registration protocols for interactive systems: a methodology and a case of study. *CoRR*, abs/1201.1134, 2012. (Cited on page 89)

[81] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Fair anonymity for the Tor network. *CoRR*, abs/1412.4707, 2014. (Cited on page 143)

[82] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. A formal methodology for integral security design and verification of network protocols. *Journal of Systems and Software*, 89:87–98, 2014. (Cited on page 33, 37, 56, 58, 60, 62, 75)

[83] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. Methodological security verification of a registration protocol. In *CISIS*, pages 214–221, 2014. (Cited on page 33, 45, 69)

[84] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. New x.509-based mechanisms for fair anonymity management. *Computers & Security*, 46:111–125, 2014. (Cited on page 93, 115)

[85] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. On securing online registration protocols: Formal verification of a new proposal. *Knowl.-Based Syst.*, 59:149–158, 2014. (Cited on page 44, 69)

[86] Jesus Diaz, David Arroyo, and Francisco B. Rodriguez. `libgroupsig`: an extensible c library for group signatures. *submitted*, 2015. (Cited on page 55)

[87] Jesus Diaz, Seung Geol Choi, David Arroyo, Angelos D. Keromytis, Francisco B. Rodriguez, and Moti Yung. e-shopping Incognito: Comprehensive Privacy-Enhanced Mechanisms for Global E-Commerce. In *In preparation*, 2015. (Cited on page 121, 197)

[88] Jesus Diaz and Antonio Sanchez. Bitcoin: A cryptographic currency, 2014. (Cited on page 136)

[89] T. Dierks and E. Rescorla. RFC5246 - the transport layer security (tls) protocol version 1.2. `http://tools.ietf.org/html/rfc5246`, 2008. (Cited on page 15, 128)

[90] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. (Cited on page 17)

[91] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association. (Cited on page 7, 15, 93, 128, 144, 145, 182)

[92] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. Security analysis of mobile two-factor authentication schemes. *Intel Technology Journal, ITJ66 Identity, Biometrics, and Authentication Edition*, 18(4), July 2014. (Cited on page 69)

[93] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983. (Cited on page 16, 17, 21, 73)

[94] John R. Douceur. The sybil attack. In *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, pages 251–260, 2002. (Cited on page 148)

[95] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 139–147, 1992. (Cited on page 148)

[96] Carl Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, 2007. `http://eprint.iacr.org/`. (Cited on page 52, 53)

[97] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas F. La Porta. Exploiting open functionality in sms-capable cellular networks. In *ACM Conference on Computer and Communications Security*, pages 393–404, 2005. (Cited on page 69, 90)

[98] ENISA. Standardisation in the field of electronic identities and trust service providers. Technical report, December 2014. (Cited on page 6)

[99] The Eurostat. E-commerce by individuals and enterprises. (Cited on page 122)

[100] Uriel Feige, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *STOC*, pages 210–217, 1987. (Cited on page 200)

[101] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003. (Cited on page 55)

[102] Simson L. Garfinkel. Email-based identification and authentication: An alternative to pki? *IEEE Security & Privacy*, 1(6):20–26, 2003. (Cited on page 8, 34, 45)

[103] Simson L. Garfinkel, David Margrave, Jeffrey I. Schiller, Erik Nordlander, and Robert C. Miller. How to make secure email easier to use. In *CHI*, pages 701–710, 2005. (Cited on page 45)

[104] Simson L. Garfinkel and Robert C. Miller. Johnny 2: a user test of key continuity management with s/mime and outlook express. In *SOUPS*, pages 13–24, 2005. (Cited on page 71)

[105] Hamza Ghani, Jesus Luna, and Neeraj Suri. Quantitative assessment of software vulnerabilities based on economic-driven security metrics. In *CRiSIS*, pages 1–8, 2013. (Cited on page 33)

[106] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001. (Cited on page 17)

[107] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996. (Cited on page 13, 93)

[108] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. (Cited on page 26)

[109] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988. (Cited on page 201)

[110] Google. google-authenticator. `http://code.google.com/p/google-authenticator/`, 2013. (Cited on page 89)

[111] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real c code. In *VMCAI*, pages 363–379, 2005. (Cited on page 35)

[112] Ryan Henry and Ian Goldberg. Extending nymble-like systems. In *IEEE Symposium on Security and Privacy*, pages 523–537, 2011. (Cited on page 96, 148)

[113] Ryan Henry and Ian Goldberg. Thinking inside the BLAC box: Smarter protocols for faster anonymous blacklisting. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2013)*. ACM, November 2013. (Cited on page 96)

[114] Ryan Henry, Kevin Henry, and Ian Goldberg. Making a nymbler nymble using verbs. In *Privacy Enhancing Technologies*, pages 111–129, 2010. (Cited on page 96)

[115] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover security design flaws using the STRIDE approach. `http://msdn.microsoft.com/en-us/magazine/cc163519.aspx`, 2006. (Cited on page 35)

[116] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Commun. ACM*, 50(1):79–83, 2007. (Cited on page 55)

[117] Alfred Horn. On sentences which are true of direct unions of algebras. *J. Symb. Log.*, 16(1):14–21, 1951. (Cited on page 18)

[118] Hanane Houmani, Mohamed Mejri, and Hamido Fujita. Secrecy of cryptographic protocols under equational theory. *Knowl.-Based Syst.*, 22(3):160–173, 2009. (Cited on page 17)

[119] IEEE. Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE Standard 802.11, June 1999. (Cited on page 34, 49, 50, 51)

[120] ISO/IEC. ISO/IEC CD 20008-1: Information technology - security techniques - anonymous digital signatures - part 1: General, 2012. (Cited on page 56, 58, 97)

[121] ISO/IEC. ISO/IEC CD 20008-2: Information technology - security techniques - anonymous digital signatures - part 2: Mechanisms using a group public key, 2012. (Cited on page 56)

[122] ISO/IEC. ISO/IEC CD 20009-1.2 draft: Information technology - security techniques - anonymous entity authentication - part 1: General, 2012. (Cited on page)

[123] ISO/IEC. ISO/IEC CD 20009-2.2 draft: Information technology - security techniques - anonymous entity authentication - part 2: Mechanisms based on signatures using a group public key, 2012. (Cited on page 97)

[124] Toshiyuki Isshiki, Kengo Mori, Kazue Sako, Isamu Teranishi, and Shoko Yonezawa. Using group signatures for identity management and its implementation. In *Proceedings of the 2006 Workshop on Digital Identity Management, Alexandria, VA, USA, November 3, 2006*, pages 73–78, 2006. (Cited on page 56)

[125] Markus Jakobsson and David M'Raïhi. Mix-based electronic payments. In *Selected Areas in Cryptography*, pages 157–173, 1998. (Cited on page 123)

[126] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. Technical Report 19, Verimag Technical Report, 2004. (Cited on page 21)

[127] Sanjeev Jha, Montserrat Guillen, and J. Christopher Westland. Employing transaction aggregation strategy to detect credit card fraud. *Expert Syst. Appl.*, 39(16):12650–12657, 2012. (Cited on page 134)

[128] Peter C. Johnson, Apu Kapadia, Patrick P. Tsang, and Sean W. Smith. Nymble: Anonymous ip-address blocking. In *Privacy Enhancing Technologies*, pages 113–133, 2007. (Cited on page 96)

[129] W.-S. Juang and C.-L. Lei. Partially blind threshold signatures based on discrete logarithm. *Computer Communications*, 22(1):73–86, 1999. (Cited on page 25)

[130] Jan Jurjens. *Secure Systems Development with UML*. SpringerVerlag, 2003. (Cited on page 35, 36)

[131] Brian W. Kernighan and Dennis Ritchie. *The C Programming Language, Second Edition*. Prentice-Hall, 1988. (Cited on page 102)

[132] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In *EUROCRYPT*, pages 571–589, 2004. (Cited on page 13, 23, 24, 56, 99, 109, 110, 111, 113, 138, 200)

[133] Manoj Kumar, Anand Rangachari, Anant Jhingran, and Rakesh Mohan. Sales promotions on the internet. In *Proceedings of the 3rd conference on USENIX Workshop on Electronic Commerce - Volume 3*, WOEC98, pages 14–14, Berkeley, CA, USA, 1998. USENIX Association. (Cited on page 134)

[134] Robert Künnemann. *Foundations for analyzing security APIs in the symbolic and computational model*. PhD thesis, École normale supérieure de Cachan, 2010. (Cited on page 35)

[135] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with xor to the xor-free case in the horn theory based approach. In *ACM Conference on Computer and Communications Security*, pages 129–138, 2008. (Cited on page 50)

[136] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. (Cited on page 16)

[137] Benoît Libert, Thomas Peters, and Moti Yung. Group signatures with almost-for-free revocation. In *CRYPTO*, pages 571–589, 2012. (Cited on page 23, 63, 111)

[138] Benoît Libert, Thomas Peters, and Moti Yung. Scalable group signatures with revocation. In *EUROCRYPT*, pages 609–627, 2012. (Cited on page 111)

[139] Zi Lin and Nicholas Hopper. Jack: scalable accumulator-based nymble system. In *WPES*, pages 53–62, 2010. (Cited on page 96)

[140] Peter Lofgren and Nicholas Hopper. Bnymble: More anonymous blacklisting at almost no cost (a short paper). In *Financial Cryptography*, pages 268–275, 2011. (Cited on page 96)

[141] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, pages 184–199, 1999. (Cited on page 201)

[142] Shin'ichiro Matsuo, Kunihiko Miyazaki, Akira Otsuka, and David A. Basin. How to evaluate the security of real-life cryptographic protocols? - the cases of iso/iec 29128 and cryptrec. In *Financial Cryptography Workshops*, pages 182–194, 2010. (Cited on page 35, 39, 42, 44, 77, 122)

[143] Ueli Maurer. Constructive cryptography - a new paradigm for security definitions and proofs. In *TOSCA*, pages 33–56, 2011. (Cited on page 17)

[144] Ueli M. Maurer. Information-theoretic cryptography. In *CRYPTO*, pages 47–64, 1999. (Cited on page 17)

[145] Damon McCoy, Kevin S. Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas C. Sicker. Shining light in dark places: Understanding the Tor network. In *Privacy Enhancing Technologies*, pages 63–76, 2008. (Cited on page 93)

[146] Catherine Meadows. The NRL protocol analyzer: An overview. *J. Log. Program.*, 26(2):113–131, 1996. (Cited on page 35)

[147] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. (Cited on page 11)

[148] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1):1–40, 1992. (Cited on page 18)

[149] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Inf. Comput.*, 100(1):41–77, 1992. (Cited on page 18)

[150] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560: X.509 internet public key infrastructure online certificate status protocol - OCSP, 1999. (Cited on page 8, 96, 99, 177, 178)

[151] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. (Cited on page 136)

[152] Toru Nakanishi, Nobuaki Haruna, and Yuji Sugiyama. Unlinkable electronic coupon protocol with anonymity control. In *ISW*, pages 37–46, 1999. (Cited on page 123)

[153] P. Neumann. Principled assuredly trustworthy composable architectures. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 2004. (Cited on page 55)

[154] nexcess. Magento Hosting: Best practices for Optimum Performance, 2013. (Cited on page 139)

[155] U.S. Department of Commerce. The 2nd quarter 2013 retail e-commerce sales report. (Cited on page 122)

[156] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In *TCC*, pages 80–99, 2006. (Cited on page 25, 200)

[157] Pierluigi Paganini and Richard Amores. *The Deep Dark Web: The Hidden World*. CreateSpace Independent Publishing Platform, 2012. (Cited on page 94)

[158] Kurt Partridge, Manas A. Pathak, Ersin Uzun, and Cong Wang. Picoda: Privacy-preserving smart coupon delivery architecture, 2012. (Cited on page 134)

[159] Larry Paulson, Tobias Nipkow, and Makarius Wenzel. Isabelle, 2012. (Cited on page 17)

[160] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998. (Cited on page 17)

[161] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991. (Cited on page 26, 138)

[162] Y. Pettersen. RFC6961 - the transport layer security (tls) multiple certificate status request extension. `http://tools.ietf.org/html/rfc6961`, 2013. (Cited on page 115)

[163] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudonymity - A proposal for terminology. In *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 1–9, 2000. (Cited on page 12)

[164] Klaus Potzmader, Johannes Winter, Daniel Hein, Christian Hanser, Peter Teufl, and Liqun Chen. Group signatures on mobile devices: Practical experiences. In *Trust and Trustworthy Computing - 6th International Conference, TRUST 2013, London, UK, June 17-19, 2013. Proceedings*, pages 47–64, 2013. (Cited on page 56, 63)

[165] Sören Preibusch, Thomas Peetz, Günes Acar, and Bettina Berendt. Purchase details leaked to PayPal (Short Paper). In *Financial Cryptography*, 2015. (Cited on page 121, 123, 140)

[166] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. How to explain zero-knowledge protocols to your children. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 628–631, 1989. (Cited on page 26)

[167] Zulfikar Ramzan. Phishing attacks and countermeasures. In *Handbook of Information and Communication Security*, pages 433–448. 2010. (Cited on page 71)

[168] James Ransome and Anmol Misra. *Core Software Security: Security at the Source*. Auerbach Publications, Boston, MA, USA, 2013. (Cited on page 4)

[169] Alfredo Rial. *Privacy-Preserving E-Commerce Protocols*. PhD thesis, Arenberg Doctoral School, KU Leuven, 2013. (Cited on page 123)

[170] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001. (Cited on page 23)

[171] Tomas Sander and Amnon Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In *Financial Cryptography*, pages 46–61, 1999. (Cited on page 123)

[172] Edward J. Schwartz, David Brumley, and Jonathan M. McCune. Contractual anonymity. In *NDSS*, 2010. (Cited on page 96)

[173] S. Sheng, L. Broderick, C.A. Koranda, and J.J. Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software, 2006. (Cited on page 71)

[174] A. Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014. (Cited on page 35)

[175] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004. (Cited on page 17)

[176] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In *EUROCRYPT*, pages 209–219, 1995. (Cited on page 24, 123)

[177] Frank Stajano. Pico: No more passwords! In *Security Protocols XIX - 19th International Workshop, Cambridge, UK, March 28-30, 2011, Revised Selected Papers*, pages 49–81, 2011. (Cited on page 90)

[178] Scott Swigart and Sean Campbell. Sdl series – article #1: Investigating the security development lifecycle at microsoft. `http://www.microsoft.com/security/sdl/resources/publications.aspx`, October 2008. (Cited on page 35)

[179] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology, 2002. (Cited on page 4)

[180] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blacklistable anonymous credentials: blocking misbehaving users without TTPs. In *ACM Conference on Computer and Communications Security*, pages 72–81, 2007. (Cited on page 96)

[181] Patrick P. Tsang, Apu Kapadia, Cory Cornelius, and Sean W. Smith. Nymble: Blocking misbehaving users in anonymizing networks. *IEEE Trans. Dependable Sec. Comput.*, 8(2):256–269, 2011. (Cited on page 96)

[182] Timothy W. van der Horst and Kent E. Seamons. Simple authentication for the web. In *SecureComm*, pages 473–482, 2007. (Cited on page xix, 71, 88, 89)

[183] Alma Whitten and J. D. Tygar. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association. (Cited on page 6, 71)

[184] Ford-Long Wong and Frank Stajano. Multi-channel protocols. In *Security Protocols Workshop*, pages 112–127, 2005. (Cited on page 71)

[185] ITU-T Recommendation X.509. Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks. Technical report, 11 2008. (Cited on page 56, 58, 62, 91, 95, 96, 97, 98, 177)

APPENDIX A

Definitions of CRL and OCSP extensions and ACFP messages

This appendix includes an ASN.1-like definition of the proposed *CRL* and *OCSP extensions*, and the *Anonymous Certificate Fairness Protocol* (*ACFP*). The definitions are not intended to be fully ASN.1 compliant, but just to give a more concise view of the extensions than what have been explained in the main body of this work.

A.1 CRL `revocationTypeInfo` extension

The proposed CRL extension **revocationTypeInfo**, is a CRL entry extension [185, Section 8.5.2]. Specifically, it is constructed as follows:

```
entryType  ::= ENUMERATED {
     normal (0)
     group (1)
     unlinkGroupMembers (2)
     anonGroupMembers (3) }
memberRevInfo  ::= CHOICE OPTIONAL {
          unlinkGroupMembersRevInfo ::= SEQUENCE {          unlinkRevInfo
          UNLINKREVINFO }
          anonGroupMembersRevInfo ::= SEQUENCE { anonRevInfo ANONREVINFO }
          }
```

A.2 OCSP `reqTypeInfo` and `rspTypeInfo` extensions

The OCSP request extension **reqTypeInfo**, which should be included in the `singleRequestExtensions` field of a request [150, Section 4.1.1] is defined as follows:

```
reqType ::= ENUMERATED {
        normal (0)
        group (1)
        unlinkGroupMembers (2)
        anonGroupMembers (3)
        unlinkability (4)
        anonymity (5)
        groupSignature (6) }
ACFPref  INTEGER OPTIONAL
reqInfo  REQUESTINFO OPTIONAL
```

The associated OCSP response extension **rspTypeInfo**, which should be included in the `singleExtensions` field of a response [150, Section 4.2.1] is defined as follows:

```
rspType ::= ENUMERATED {
        normal (0)
        group (1)
        unlinkGroupMembers (2)
        anonGroupMembers (3)
        unlinkability (4)
        anonymity (5)
        groupSignature (6) }
rspStatus ::= CHOICE OPTIONAL {
        "good"
        "unknown"
        "unlinkRevoked"
        "anonRevoked"
        "revoked"}
memberRevInfo  ::= CHOICE OPTIONAL {
        unlinkGroupMembersRevInfo ::= SEQUENCE {          unlinkRevInfo
        UNLINKREVINFO }
        anonGroupMembersRevInfo ::= SEQUENCE { anonRevInfo ANONREVINFO }
        unlinkRevInfo UNLINKREVINFO
        anonRevInfo ANONREVINFO }
```

A.3 Definition of ACFP

ACFP requests, named **tbsACFPRequest** have the following structure:

```
version  INTEGER
requester  GeneralName
evidences  ::= SEQUENCE {
     ACFPEvidenceID  ::= CHOICE {
             NULL
             INTEGER }
     policyID  INTEGER
     reqType  ::= ENUMERATED {
             evidence (0)
             unlinkability (1)
             anonymity (2) }
     evidence  EVIDENCE
     singleEvdExtension  SINGLEEVDEXTENSION OPTIONAL }
reqExtensions REQUESTEXTENSION OPTIONAL
signature SIGNATURE OPTIONAL
```

And the associated responses, named **tbsACFPResponse**, are structured as follows:

```
version
rspStatus  ::= CHOICE {
     "Malformation"
     "UnsupportedVersion"
     "sigRequired"
     "OK"}
responses  ::= SEQUENCE {
     ACFPEvidenceID  INTEGER
     evRspStatus  ::= CHOICE {
         "Malformation"
         "UnknownID"
         "UnknownPolicy"
         "OK"
```

```
        "evidence"
        "denied"
        "accepted"}
    singleRspExtensions  SINGLERSPEXTENSION OPTIONAL }
rspExtensions  RESPONSEEXTENSION OPTIONAL
signature  SIGNATURE OPTIONAL
```

APPENDIX B

Security analysis and proofs for Caduceus

Next, we apply the methodology shown in Chapter 3 to the protocols in Caduceus described in Chapter 7. Moreover, in the second stage of the methodology, we include both formal and computational approaches. This exercise, besides being necessary for ensuring that the required security properties are achieved, serves us to exemplify the usefulness of combining both approaches, since each one provides additional guarantees.

In this verification process we focus ourselves in the main parts of the system. That is: turn-retrieval and checkout. As we mentioned in Section 7.6.1, being this a setting where high security is necessary, it is unavoidable to require physical presence of the customer when at least once when first joining the system, although, after that, the protocol in Chapter 5 may be incorporated. However, since we already formally proved the security of this protocol in Chapter 5, we omit it here.

Like we did for the registration protocol SEBIA in Chapter 5, for the sake of brevity, we initiate the methodology having a design candidate, which is the system introduced in Chapter 7.

B.1 Informal analysis

The first task is to perform the initial informal analysis. The five steps comprising this stage are as follows.

Step 1. Goals of the system. The goals correspond to the privacy and unforgeability security requirements described in Section 7.5.

Step 2. Attacker capabilities and security model. For the formal verification, we adopt the Dolev-Yao model. In the computational verification we will grant the attacker additional capabilities, like the ability to corrupt customers. The latter are defined in Section B.3.

Step 3. Entities assumptions, design candidate and channels abstraction.

- **Design candidate**. In this case we start from the design depicted in Figure 7.3 and described in detail in Section 7.3. For the sake of clarity, we summarize the

transmitted messages in the sequence diagrams in Figure B.1 (turn-retrieval) and Figure B.2 (checkout) .

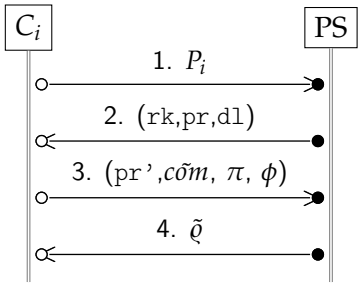


Figure B.1: Sequence diagram for turn-retrieval in Caduceus. See Section 7.3 for more details on how is each token produced.

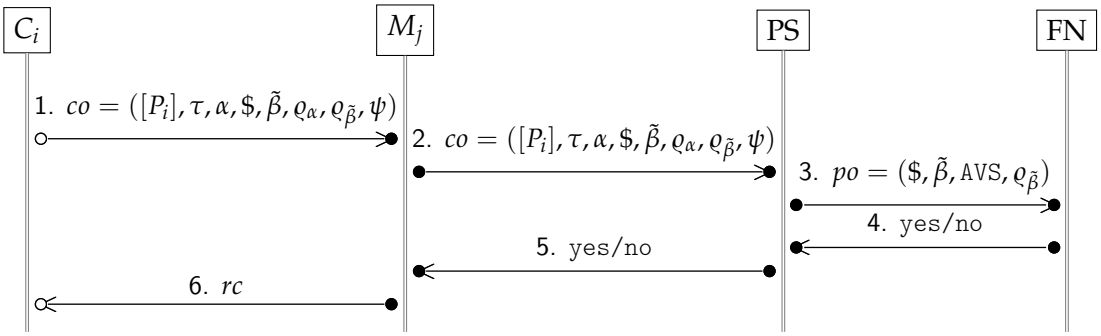


Figure B.2: Sequence diagram for checkout in Caduceus. See Section 7.3 for more details on how is each token produced.

- **Entities assumptions**. The entities taking part in the system may act as follows:
  - Customers are potentially dishonest. Thus, they may collude with any other entity.
  - Merchants are also potentially dishonest and may collude with any other entity.
  - PS acts as a semi-honest party but can collude with everyone.
  - FN is a semi-honest party and cannot collude with any other entity.

- **Communication channels**. All the communications performed during the protocols in the system are assumed to be confidential. Also, merchants, PS and FN authenticate their end in a conventional manner (i.e., with conventional digital certificates).

  In addition, all communications originated by a customer must be routed through sender-anonymous channels, like those provided by the Tor network [91]. Customer (anonymous) authentication must be provided by the system (i.e., not by the communication channels).

Step 4. Informal requirements. Table B.1 shows the result obtained step by step for the turn-retrieval protocol and Table B.2 shows the result for the checkout process. For brevity, we omit the checks performed between the different message exchanges and assume that if a message at step n is sent, then all the previous checks have been successful. We refer to Section 7.3 for the detailed processes. In this analysis, besides authenticity and confidentiality, we add properties for anonymity and pseudonymity with respect to a token issued by an entity E, that will be denoted An_E and Pn_E, respectively. Note that these are special types of authenticity.

Step in Figure B.1	P_i	rk,pr,dl	pr',$\tilde{\beta}$,π,ϕ	$\tilde{\varrho}$
1. $C_i \rightarrow$ PS : P_i	none			
2. PS $\rightarrow C_i$: rk,pr,dl	none	A_{PS}		
3. $C_i \rightarrow$ PS : (pr',$\tilde{\beta}$,π,ϕ)	none	A_{PS}	none	
4. PS $\rightarrow C_i$: $\tilde{\varrho}$	C, Pn_{C_i}	$A_{\mathsf{PS}}, Pn_{C_i}, C$	$A_{\mathsf{PS}}, Pn_{C_i}, C$	$A_{\mathsf{PS}}, Pn_{C_i}, C$

Table B.1: Informal requirements table for the turn-retrieval protocol. The rows under column *Step in Figure A.1* reference the corresponding step in Figure B.1.

Step in Figure B.2	co	$po_{\$,\mathtt{AVS}}$	$po_{\tilde{\beta},\varrho_{\tilde{\beta}}}$	yes	rc
1. $C_i \rightarrow M_j$: co	none				
2. $M_j \rightarrow$ PS : co	none				
3. PS $\rightarrow$ FN : po	none	none	none		
4. FN $\rightarrow$ PS : yes	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	$A_{\mathsf{PS},\mathsf{FN}}, C$	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	A_{FN}, C	
5. PS $\rightarrow M_j$: yes	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	$A_{\mathsf{PS},\mathsf{FN}}, C$	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	$A_{\mathsf{FN},\mathsf{PS}}, C$	
6. $M_j \rightarrow C_i$: rc	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	$A_{\mathsf{PS},\mathsf{FN}}, C$	An_{C_i}/Pn_{C_i}, $A_{M_j,\mathsf{PS},\mathsf{FN}}, C$	$A_{\mathsf{FN},\mathsf{PS}}, C$	$A_{\mathtt{M}_j}, C$

Table B.2: Informal requirements table for the checkout protocol. The rows under column *Step in Figure A.2* reference the corresponding step in Figure B.2. For messages composed by several components (e.g., po), we use subindexes to specify subcomponents that require different properties. E.g., $po_{\tilde{\beta}}$ refers to the component $\tilde{\beta}$ of po. Note that we assume that yes is sent in steps 4 and 5; otherwise, the process stops.

For the formal verification of this properties, in this case the two approaches are necessary. That is, we use the formal approach, with ProVerif, for ensuring that the protocol keeps the required informal requirements as stated in the previous tables. In this case, this approach guarantees that no intentional manipulation of the messages may cause a security flaw. Moreover, this protocols have a high load of cryptographic primitives, including complex relationships between the different tokens produced during the protocols. Thus, even though we include abstractions of them in our ProVerif models, the Dolev-Yao model does not provide enough guarantees, and a computational verification is needed.

Therefore, for the formal verification, we use ProVerif, since it allows to verify both confidentiality, authenticity and anonymity (through observational equivalence) properties. The computational verification is described in Section B.3.

Step 5. Informal verification. We proceed, for each protocol, to informally verify the properties assigned to the message components at step 4, considering the attacker model and the channels abstraction produced at steps 2 and 3, respectively.

Turn-retrieval protocol :

Message 1. P_i. We expect the customer's pseudonym to be kept confidential and to authenticate her pseudonymously. However, at this point we cannot assume any of these properties, since as far as PS is concerned, the message may come from a dishonest customer (or directly from the attacker). Thus, we place none requirement in the tokens at this point.

Message 2. `rk,pr,dl`. The three tokens (risk, promotions and deadline, respectively) are produced and sent by PS after searching P_i in its database. Therefore, they should provide authenticity on the side of PS. Still, we cannot expect confidentiality, for the same reason as in Message 1. Since the channel provides authenticity on the side of PS, this is correct.

Message 3. `pr'`, β, π, ϕ. As in Message 1, we cannot place any requirement on this message, since it may come from a dishonest user or the attacker.

Message 4. $\tilde{\varrho}$. If this message is sent, means that all the checks performed by the cryptographic operations have succeeded. Thus, the customer must be the legitimate owner of P_i, and all the previously sent messages "inherit" the pseudonymity authentication requirement and also confidentiality. Additionally, the message $\tilde{\varrho}$ is also assumed to be authenticated by PS, since it originates from it. Given the properties of the underlying cryptographic primitives, and those of the communication channels, all properties are satisfied.

Checkout protocol :

Message 1. *co*. As before, we cannot assume anything about this message, since it may come from the attacker.

Message 2. *co*. Since, for checkout, M_j may be acting dishonestly, we cannot place any requirement at this step either.

Message 3. *po*. PS might also be behaving dishonestly during checkout. Therefore, following the same reasoning as in Message 2, we cannot place any requirement on *po* at this step, nor modify those of *co*.

Message 4. `yes`. If FN accepts the previous cryptographic tokens, answering with a `yes`, it means that they are correct. Hence, *co* is required to be anonymously or pseudonymously authenticated at the customer's side (depending on whether she chose to checkout anonymously or pseudonymously). Furthermore, since FN received the information directly from PS, and the channel requires authenticity by PS, all the previous tokens inherit also authenticity by PS. A similar reasoning applies to M_j, given that we already know at this point that PS is acting honestly (or FN has evidence enough to make it liable). Additionally, for the same motives, confidentiality may also be required. As for the `yes` message itself, it is confidential (since PS is honest) and authenticated by FN. Given that all the communication channels involved up to this point are assumed to provide authenticity (or anonymity/pseudonymity for the customer) and confidentiality, ensures that all the

properties are kept. As for the components within po that originated from C_i ($po_{\tilde{\beta},\varrho_{\tilde{\beta}}}$), they receive the same requirements as co. $po_{\$,\texttt{AVS}}$, that was included by PS, also receives the same except A_{M_j}. For the same reasons as above, these properties are kept.

Message 5. `yes`. This message does not modify the previous requirements further than ensuring authenticity by PS to the `yes` message. Since PS is known to be acting honestly at this point, and the involved channel ensures confidentiality, and authenticity on PS side, the requirement is met.

Message 6. rc. When the merchant M_j sends this message (and after the customer verifies it, step omitted in Table B.2 but assumed here), we may require authenticity (on the merchant's side) and confidentiality on it, since all steps are assumed to have been run correctly. Given that all entities are trusted to be behaving honestly, and the channels meet all the requirements, there is no inconsistency up to this point.

After performing the informal analysis of the protocols in Caduceus, we have not found any design flaw. Therefore, we may proceed to the formal analysis phase, which in this case will be performed in the following sections using both the formal model and the computational model.

B.2 Procedural verification: formal model

In this section, we formally verify the system, following the informal verification phase. However, rather than including an exhaustive account of all the properties to be verified that were obtained in the informal phase, we focus on the most important ones, and in introducing the equational theory employed for group signatures, partially blind signatures and the other cryptographic primitives used in the system. Nevertheless, all the properties are verified in the ProVerif source code available online[1], and the pseudocode, required as the initial step of this phase, corresponds to the processes described in Section 7.3.

Note also that, while the computational verification is included in Section B.3, here we additionally try to determine whether or not an attacker would be able to break them by sniffing the communications, replaying or modifying messages, or somehow altering the legitimate behavior of the protocol. The combination of the guarantees provided by both approaches certainly ensures a higher level of security.

Finally, throughout this section, and in order to ease the explanation of the code snippets that will be included, we temporarily drop the notation used in the preceding chapters with respect to cryptographic functionality and information tokens. We now proceed to explain the model for ProVerif, including these temporary notation.

B.2.1 ProVerif model

The entities, channels and overall process have already been defined in Section 7.3. Here, we describe how we have modeled it for ProVerif[2].

[1] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

[2] The complete source code is available at `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`.

Equational theory. Next, we define the equational theory, in terms of constructors and destructors, for allowing ProVerif to reason about the cryptographic primitives that take part in our system. Below, constructors are denoted with conventional teletype font, and include types for its variables. Destructors are denoted with bold teletype font, and for readability, we do not specify the types of its variables, since it will be clear from the context.

Group signatures. The types defined for the primitives related to group signatures are *MgrKey*, *GrpKey* and *MemKey* for manager, group and member keys, respectively. Member keys are associated to a name[3] of type *Id*, and group signatures are of type *bitstring*.

$$
\begin{array}{lcl}
\mathtt{gs}_{getgrpM}(mgr : MgrKey) & = & GrpKey \\
\mathtt{gs}_{join}(id : Id, grp : GrpKey, mgr : MgrKey) & = & MemKey \\
\mathtt{gs}_{sign}(m : bitstring, mem : MemKey, grp : GrpKey) & = & bitstring \\
\mathbf{gs_{verify}}(m, & & \\
\quad \mathtt{gs}_{sign}(m, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr), \mathtt{gs}_{getgrpM}(mgr)), & & \\
\quad \mathtt{gs}_{getgrpM}(mgr)) & = & true \\
\mathbf{gs_{open}}(\mathtt{gs}_{sign}(m, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr), \mathtt{gs}_{getgrpM}(mgr)), & & \\
\quad mgr, & & \\
\quad \mathtt{gs}_{getgrpM}(mgr)) & = & id
\end{array}
$$

Note that we only define the *join*, *sign*, $verify$ and $open$ functions. The group creation functionality (setup) would be implicitly performed by creating the manager key (from which the rest of the keys are derived). Additionally, we define a helper function *getgrpM* that allows to obtain the group key associated to a given manager key. As for the claiming-related functionality, it is not implemented in the model, since we focus the analysis on the two main phases: turn retrieval and checkout.

Partially Blind Signatures. For this primitive, we use *PbsPrvKey* and *PbsPubKey* as types for private and public keys, and use variables of type *Nonce* for blinding and unblinding. The different tokens (blinded messages, signatures and proofs) are all of type *bitstring*.

$$
\begin{array}{lcl}
\mathtt{pbs}_{getpub}(prv : PbsPrvKey) & = & PbsPubKey \\
\mathtt{pbs}_{blind}(msg : bitstring, cmn : bitstring, n : Nonce, pub : PbsPubKey) & = & bitstring \\
\mathbf{pbs_{blindmsg\_verify}}(cmn, & & \\
\quad \mathtt{pbs}_{blind}(msg, cmn, n, \mathtt{pbs}_{getpub}(prv)), & & \\
\quad prv) & = & true \\
\mathtt{pbs}_{sign}(cmn : bitstring, bmsg : bitstring, prv : PbsPrvKey) & = & bitstring \\
\mathbf{pbs_{blindsig\_verify}}(msg, cmn, n, & & \\
\quad \mathtt{pbs}_{sign}(c, & & \\
\quad\quad \mathtt{pbs}_{blind}(msg, cmn, n, \mathtt{pbs}_{getpub}(prv)), & & \\
\quad\quad prv), \mathtt{pbs}_{getpub}(prv)) & = & true \\
\mathtt{pbs}_{unblind}(bsig : bitstring, n : Nonce, pub : PbsPubKey) & = & bitstring \\
\mathbf{pbs_{verify}}(msg, cmn, & & \\
\quad \mathtt{pbs}_{unblind}(\mathtt{pbs}_{sign}(cmn, \mathtt{pbs}_{blind}(msg, cmn, n, \mathtt{pbs}_{getpub}(prv)), prv), & & \\
\quad\quad n, \mathtt{pbs}_{getpub}(prv)), & & \\
\quad \mathtt{pbs}_{getpub}(prv)) & = & true
\end{array}
$$

[3] A name is an atomic value in ProVerif, similar to a constant in a "conventional" programming language.

Commitments. In the context given by our online shopping proposal, commitments are made over a group member key, using a nonce to blind it. Being the member key what is being committed, we do not include the open functionality, since it would de-anonymize the customer, and we are just interested in making the customer requesting the turn prove that he is the same than will later use it during checkout.

$$\mathtt{commit}(n : Nonce, mem : MemKey) \quad = \quad bitstring$$

Zero-knowledge proofs. Zero-knowledge proofs are used to demonstrate equality of the secrets included within commitments (which are, in turn, the blindly signed messages of the partially blind signatures) and group signatures. They do not require new types definitions.

$$\begin{array}{lll}
\mathtt{zkpeq}_{prove}(comm : bitstring, gs_1 : bitstring, & & \\
\qquad gs_2 : bitstring, mem : MemKey) & = & bitstring \\
\mathbf{zkpeq_{verify}}(\mathtt{zkpeq}_{prove}(commit(n, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr)), & & \\
\qquad \mathtt{gs}_{sign}(m_1, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr), \mathtt{gs}_{getgrpM}(mgr)), & & \\
\qquad \mathtt{gs}_{sign}(m_2, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr), \mathtt{gs}_{getgrpM}(mgr)), & & \\
\qquad \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr)) & & \\
\qquad \mathtt{gs}_{getgrpM}(mgr)) & = & true
\end{array}$$

Pseudonyms. We recall that a pseudonym for a given customer is implemented as a group signature on a random message created using his membership key. Thus, the functions related to pseudonyms are basically those of group signatures. Additionally, we define the type *Pseudonym* for managing pseudonyms.

$$\begin{array}{lll}
\mathtt{pnym}_{get}(n : Nonce, mem : MemKey) & = & Pseudonym \\
\mathbf{pnym_{get_nonce}}(\mathtt{pnym}_{get}(n, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr))) & = & n \\
\mathbf{pnym_{verify}}(n, & & \\
\qquad \mathtt{pnym}_{get}(n, \mathtt{gs}_{join}(id, \mathtt{gs}_{getgrpM}(mgr), mgr)), & & \\
\qquad \mathtt{gs}_{getgrpM}(mgr)) & = & true
\end{array}$$

Communication channels. As has already been pointed out, we assume that SSL/TLS is used over some anonymizing network (like Tor). For keys distribution, instead of modeling it like we did in Chapter 5, which would certainly make the model more complex, we just make the session initiator create a symmetric key and securely send it to the receiver via a private channel. Note that this is the opposite approach to the one we used in Chapter 5, where we assumed that the attacker was able to initiate SSL sessions impersonating a registering user. Here this is not the case, since the customer uses from the beginning her member key to authenticate (via zero-knowledge proofs). Therefore, if the attacker obtains a member key, then it behaves exactly as the legitimate customer. However, we recall that, by assumption, these member keys are securely distributed. Hence, the attacker is not allowed to *steal* member keys and use them to attack the required properties. Still, this kind of dishonest members will be dealt with in the computational analysis performed in Section B.3, when taking into account possible collusion scenarios.

```
1  (** The customer process. **)
2  (** Receives the symmetric key for communicating with PS (note that,
3      in practice, it could just be negotiated using SSL **)
4  let customer (pbspub : PbsPubKey, pubfn : PubKey, grp : GrpKey) =
5
6          (* Creates a member key *)
7          new id : Id;
8          out (prvjoin, (msgjoin1, id));
9          in (prvjoin, (=msgjoin2, mem:MemKey));
10
11         (* Creates the pseudonym *)
12         new npnym: Nonce;
13         let pnym = pnym_get(npnym, mem) in
```

Figure B.3: Customer process for ProVerif: initialization.

Entities modeling. We create one entity per party involved in the protocol, plus a group manager process in charge of adding new group members. We separate this functionality in order to allow arbitrary new member additions without forcing it to happen at the beginning of the FN process. Customers receive as parameter the group key, the public key of FN, and the public key for partially blind signatures. PS receives the private key for partially blind signatures and the group key. FN receives the group manager key and its private key. Note that merchants do not receive any keying material as parameter. This is because they basically act as forwarders in this simple model. In the listings included below, showing the code of each entity, the functions that have been described above are written (e.g., for the group sign function) `gs_sign` instead of $\texttt{gs}_{sign}$, following the notation used in the source code available online[4].

Customers process. We show the code for the customer process in three separate listings. Figure B.3 shows the initialization. Specifically, it is worth noting how the member key is created (lines 7 through 9, included). The customer first creates a fresh identity (`id` variable) and sends it over the private channel `prvjoin`, which is only used for adding new members. This message will be received by the process `gm`, the Group Manager, which basically executes the $\texttt{gs}_{join}$ and returns the result via `prvjoin`. Afterwards, she creates her pseudonym (lines 12 and 13) using the received member key and a fresh nonce, `npnym`.

Figure B.4 shows the first half of the turn retrieval phase. Concretely, the customer sends her pseudonym to PS (line 6), via the public channel, but encrypted with the secret key `kcps`, shared with PS ("negotiated" in lines 2 and 3). As a result, the customer receives from PS a risk factor `rk`, set of promotions `pr` and deadline `dl` (lines 9 and 10).

In the second half of the turn retrieval process, depicted in Figure B.5, customer and PS jointly produce a partially blind signature, where the common message will be the tuple (`rk`,`pr`,`dl`) and the blindly signed message a commitment to the customer's member key. First, the customer commits to her member key in line 5, and then blinds the result invoking `pbs_blind` in line 8. Subsequently, the customer creates a proof showing in zero-knowledge the correctness of the blinded message. The blinded message and the ZK proof are sent to PS in line 14, along

[4] `htpp://www.ii.uam.es/~gnb/thesis-jdv.tgz`

```
1   (* Symmetric key shared with PS - e.g. negotiated using SSL *)
2   new kcps : Key;
3   out (prvnetcps, (msgprv1, kcps));
4
5   (* Sends to PS: pnym, for initiating the turn retrieval *)
6   out (net, senc((msg1, pnym), kcps));
7
8   (* Receives from PS: risk, promos, deadline *)
9   in (net, cmsg2:bitstring);
10  let (=msg2, rk:bitstring, pr:bitstring, dl:bitstring) =
11                                          sdec(cmsg2, kcps) in
```

Figure B.4: Customer process for ProVerif: first half of turn retrieval.

with the chosen promotions (in this model, the customer always sends `pr` back, which may be interpreted as if she always accepts all promotions). After receiving and decrypting the response from PS (lines 17 and 18), the customer verifies the blinded signature by calling `pbs_blindsig_verify` (line 21) and, if the verification succeeds, unblinds it with `pbs_unblind`, using the same nonce that was used for blinding the message (`n2`, created in line 2).

The customer behavior during checkout is shown in Figure B.6. This phase begins with the customer issuing group signatures of the purchase information (lines 2 and 3), the payment information (lines 6 and 7) and creating a zero-knowledge proof showing that both group signatures have been issued using the same member key (line 10). Note that we simplify the payment information by making it depend on the member key (which in turn depends on the customer's real identity). This models the fact that, in case this information was learned by the attacker, she will also learn the customer's real identity. Additionally, this information is encrypted with FN's public key (also in line 6). These tokens are sent to the merchant (line 18), along with the turn obtained during the turn retrieval phase. After several messages between the merchant, PS and FN, the customer will be sent the checkout receipt (lines 22 and 23). The customer then verifies the receipt in lines 26 and 27 and ends. The final lines are just to test reachability, i.e., that this part of the code actually executes, preventing non-termination due to typos or other coding errors (and ensuring that the last event is also executed).

Merchants process. In this model, the merchant process is quite simple. Actually, it basically forwards the information received during checkout from the customer, and sends it back, signed, when the payment is executed by FN. In the real world, instead than just forwarding it, the merchant may add some additional tokens, like the AVS value (see Section 7.3.3). However, this additional information does not actually alter the information flow, and we may ignore it. Therefore, for readability, we omit the merchants process. The complete source may is available online[5].

PS process. Figure B.7 shows the code for PS during turn retrieval. Initially, PS receives a pseudonym from a customer willing to obtain a turn (lines 7 and 8). It then fetches the promotions and risk factor associated to that pseudonym (here, we just create fresh names for these values), as it is done for the turn deadline.

[5] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

```
1   (* Sends to PS: blinded signature (commitment and ZK proofs) *)
2   new n1 : Nonce; new n2 : Nonce;
3
4   (* Commit to the member key *)
5   let com = commit(n1,mem) in
6
7   (* Blindly sign the commitment *)
8   let blindcom = pbs_blind(com, (rk,pr,dl), n2, pbspub) in
9
10  (* Get the ZK proof of correctness of the blinded message *)
11  let zkblindcom = turn_getzkp(blindcom,npnym,pnym,grp,mem,n1,n2) in
12
13  event CustomerRequestsTurn(pnym, mem, rk, pr, dl);
14  out (net, senc((msg3, pr, blindcom, zkblindcom), kcps));
15
16  (* Receive from PS: turn *)
17  in (net, cmsg4:bitstring);
18  let (=msg4, blindsig:bitstring) = sdec(cmsg4, kcps) in
19
20  (* Verify blind signature *)
21  if pbs_blindsig_verify(com,(rk,pr,dl),n2,blindsig,pbspub) = true then (
22
23    event CustomerAcceptsTurn(pnym, mem, rk, pr, dl);
24
25    (* Unblind the blind signature to produce the final turn *)
26    let turn = pbs_unblind(blindsig, n2, pbspub) in
```

Figure B.5: Customer process for ProVerif: second half of turn retrieval.

These values, which will compose the common message of the partially blind signature, are sent to the customer in line 12. If the customer agrees to these values (which she always does in our model), PS will receive the blinded message in line 15. Subsequently, the zero-knowledge proof sent to prove its correctness is verified (line 20) and finally the blinded message itself is checked (line 24). If these verifications succeed, PS blindly signs the received message, running `pbs_sign` and sends the result back to the customer (lines 27 and 29, respectively).

Figure B.8 shows the verifications performed by PS during checkout. After receiving the checkout order in line 6 (and decrypting it in line 7), PS verifies the group signatures corresponding to the purchase and encrypted payment information (lines 12 and 14). Subsequently, it verifies the zero-knowledge proof showing that both group signatures and the commitment included associated to the turn (variable `com`) have all been issued by the same customer (lines 17 and 18). Finally, the turn, which is actually a partially blind signature, is verified in lines 21 and 22. There, the message that was blindly signed is `com` and the common message was the tuple `(rk, pr, dl)`.

When the verification succeeds, PS sends a payment order to FN as shown in Figure B.9. Just after sending the actual order (line 5), PS receives from FN the result of the operation (line 8). If it has succeeded (in our model, if the execution arrives at this point, we assume it has), then PS informs the merchant (lines 14 and 15).

FN process. FN is also a simple process. Upon receiving a payment order from PS

```
1     (* Create purchase item and group sign it *)
2     new pur : bitstring;
3     let gs_pur = gs_sign(pur, mem, grp) in
4
5     (* Encrypt banking information with FN's public key
6        and group sign it *)
7     let cbank = aenc(banking(mem), pubfn) in
8     let gs_cbank = gs_sign(cbank, mem, grp) in
9
10    (* Issue ZK proof of equality of group signatures *)
11    let zkpeq = zkpeq_prove(com, gs_pur, gs_cbank, mem) in
12
13    (* Create symmetric key for communications with Merchant *)
14    new kcm : Key;
15    out (prvnetcm, (msgprv2, kcm));
16
17    event CustomerInitCheckout(pnym, turn, com, pur, cbank,
18                               gs_pur, gs_cbank, zkpeq);
19    out (net, senc((msg5, com, (rk, pr, dl), turn,
20              pur, gs_pur, cbank, gs_cbank, zkpeq), kcm));
21
22    (* Receive checkout receipt *)
23    in (net, cmsg10:bitstring);
24    let (=msg10,pubm:PubKey,receipt:bitstring) = sdec(cmsg10,kcm) in
25
26    (* Verify the receipt *)
27    if verify(receipt, pubm,
28              (turn, pur, cbank, gs_pur, gs_cbank, zkpeq)) = true then (
29
30      event CustomerAcceptsReceipt(pnym, turn, com, pur, cbank,
31                                   gs_pur, gs_cbank, zkpeq);
32
33      (* Dummy message -- for testing reachability *)
34      new dummy : bitstring;
35      out (net, senc(dummy, kcm))
36
37    )
38
39  ).
```

Figure B.6: Customer process for ProVerif: checkout.

```
let ps (pbsprv : PbsPrvKey, grp : GrpKey) =

(* Receive from customer a shared key  *)
in (prvnetcps, (=msgprv1, kcps : Key));

(* Receive a pseudonym from a customer initiating turn retrieval *)
in (net, cmsg1:bitstring);
let (=msg1, pnym:Pseudonym) = sdec(cmsg1, kcps) in

(* Creates risk, promos and deadline, and sends them *)
new rk:bitstring; new pr:bitstring; new dl:bitstring;
out (net, senc((msg2, rk, pr, dl), kcps));

(* Receives blinded signature (commitment and ZK proofs) *)
in (net, cmsg3:bitstring);
let (=msg3, =pr, blindcom:bitstring, zkpblindcom:bitstring) =
                                                  sdec(cmsg3, kcps) in

(* Verify ZK proof of correctness *)
if turn_verzkp(blindcom,zkpblindcom,pnym_get_nonce(pnym),
                                    pnym,grp) = true then (

  (* Verify blinded message *)
  if pbs_blindmsg_verify((rk, pr, dl),blindcom,pbsprv) = true then (

              (* Issue blind signature and send it*)
              let blindsig = pbs_sign((rk, pr, dl),blindcom,pbsprv) in
              event PSIssuesTurn(pnym, rk, pr, dl);
              out (net, senc((msg4, blindsig), kcps));
```

Figure B.7: PS process for ProVerif. Turn retrieval phase.

```
    (* Receive symmetric key from Merchant — Simulates SSL *)
    in (prvnetmps, (=msgprv3, kmps:Key));

    (* Receive the checkout order. Here, pr might be different to
       the previous one sent before, but we assume it is the same *)
    in (net, cmsg6:bitstring);
    let (=msg6,com:bitstring,(=rk, =pr, =dl),turn:bitstring,
         pur:bitstring,gs_pur:bitstring,cbank:bitstring,
         gs_cbank:bitstring, zkpeq:bitstring) = sdec(cmsg6, kmps) in

    (* Verify the group signatures *)
    if gs_verify(pur, gs_pur, grp) = true then (

      if gs_verify(cbank, gs_cbank, grp) = true then (

        (* Verify the ZK proof of equality *)
        if zkpeq_verify(zkpeq,gs_pur,
                        gs_cbank,grp) = true then (

          (* Verify the turn (Remember: it is a blind signature) *)
          if pbs_verify(com, (rk, pr, dl), turn,
                        pbs_getpub(pbsprv)) = true then (
```

Figure B.8: PS process for ProVerif. Checkout phase: verification.

```
1            (* Send the payment info to FN *)
2            new kpsfn:Key;
3            out(prvnetpsfn, (msgprv4, kpsfn));
4
5            out(net, senc((msg7, cbank, gs_cbank), kpsfn));
6
7            (* Receive the payment confirmation *)
8            in(net, cmsg8:bitstring);
9            let (=msg8, =cbank) = sdec(cmsg8, kpsfn) in
10
11           event PSacceptsCheckout(turn,com,pur,cbank,gs_pur,
12                                   gs_cbank,zkpeq);
13
14           (* Send ACK to merchant *)
15           out(net, senc((msg9,turn,pur,cbank,
16                          gs_pur,gs_cbank,zkpeq), kmps))
17         )
18       )
19     )
20   )
21 )
22 ).
```

Figure B.9: PS process for ProVerif. Checkout phase: execution.

(line 8), it initially verifies the group signature corresponding to the encrypted payment information (line 12). If it succeeds, it then decrypts the payment information (line 15), from which it can extract the identity of the payer. This identity is compared with the result of `gs_open` in line 19. If both identities are the same (and the customer has enough funds, etc., which is ignored here), then the payment is executed. At last, FN informs PS of the outcome of this operation (line 29) and terminates.

B.2.2 Formal security proofs

Using the model explained above, the authenticity and privacy properties verified with ProVerif are as follows.

Authenticity. We show authenticity of the system in three stages. First, we show that the created turns are authentic, which gives legitimacy to the turn-retrieval phase. Then, we show authenticity of the checkout process jointly with authenticity of the generated receipts. These properties will be proven using ProVerif's correspondence assertions. As we saw in Chapter 3 and Chapter 5, these assertions are used to prove that some specific action has not been manipulated by the attacker (who cannot execute *events*).

The events defined to prove turn authenticity are as follows[6]:

- `event CustomerRequestsTurn(pnym,mem,rk,pr,dl).`
  This event signals that a customer, owner of pseudonym `pnym` and member key `mem`, has requested a turn associated to the terms specified with the risk factor `rk`, promotions `pr` and deadline `dl`.

[6]We use descriptive variable names instead of types for readability.

```
(** The FN process **)
let fn ( prvfn : PrvKey, mgr : MgrKey ) =
  (* Receive a symmetric key from PS -- simulates SSL *)
  in (prvnetpsfn, (=msgprv4, kpsfn : Key));

  (* Receive payment order *)
  in (net, cmsg7:bitstring);
  let (=msg7,cbank:bitstring,gs_cbank:bitstring) =
                                        sdec(cmsg7,kpsfn) in

  (* Verify group signature *)
  if gs_verify(cbank, gs_cbank, gs_getgrpM(mgr)) = true then (

    (* Decrypt the banking information *)
    let bank = adec(cbank, prvfn) in

    (* Check that the issuer of the group signature and the owner of
       the banking account are the same *)
    if gs_open(gs_cbank,mgr,gs_getgrpM(mgr)) = banking_getid(bank) then (

    (* Additionally, the payment amount in cbank and the one sent
       separately by PS (eliminated here) should match, but we ignore
       it in this model: see original paper for details *)

      event FNacceptsPayment(cbank, gs_cbank);

      (* Everything OK. Here FN would transfer the appropriate funds
         from the customer to the PS *)
      out (net, senc((msg8, cbank), kpsfn))

    )

).
```

Figure B.10: FN process for ProVerif.

- `event PSIssuesTurn(pnym,rk,pr,dl).`
  Indicates that PS has issued a turn, intended for the owner of the pseudonym `pnym`, and associated to the conditions stated in `(rk,pr,dl)`.

- `event CustomerAcceptsTurn(pnym,mem,rk,pr,dl).`
  Shows that a customer, owner of `pnym` and `mem`, has accepted the turn associated to `(rk,pr,dl)`.

With respect to checkout, the events used for the correspondence assertions are:

- `event CustomerInitCheckout(pnym,turn,com,pur,cbank,gs_pur,gs_cbank, zkpeq).`
  Indicates that a customer with pseudonym `pnym` has initiated a checkout using turn `turn`, for commitment `com` and purchase `pur` with group signature `gs_pur`, where the encrypted payment information is `cbank` and its group signature is `gs_cbank`. The zero-knowledge proof showing equality of the group signatures and the commitment is `zkpeq`.

- `event FNacceptsPayment(cbank,gs_cbank).`
  Is the event for showing that FN has accepted the payment order related to the payment object `cbank`, with group signature `gs_cbank`.

- `event PSacceptsCheckout(turn,com,pur,cbank,gs_pur,gs_cbank,zkpeq).`
  Signals the event in which PS accepts the payment for the same tokens as in `CustomerInitCheckout`, except the pseudonym, since we are in an anonymous checkout.

- `event MerchantIssuesReceipt(turn,com,pur,cbank,gs_pur,gs_cbank,zkpeq).`
  It is the equivalent to `PSacceptsCheckout`, although at the merchant's side, which also implies that the merchant has issued the associated receipt.

- `event CustomerAcceptsReceipt(pnym,turn,com,pur,cbank,gs_pur,gs_cbank, zkpeq).`
  Means that the customer has accepted the receipt associated to the previously defined tokens.

Turn authenticity. This property is proven with the following correspondence assertion:

```
inj-event(CustomerAcceptsTurn(pnym, mem, rk, pr, dl))
  ==> (inj-event(PSIssuesTurn(pnym, rk, pr, dl))
       && inj-event(CustomerRequestsTurn(pnym, mem, rk, pr, dl))).
```

That is, the turn will be authentic if the event by means of which a Customer accepts a turn associated to the tokens specified in the left hand side of the implication is always preceded by:

- The event with which PS signals that it has issued a turn for the same tokens.
- And the event with which the customer has requested a turn associated to the same tokens.

Specifically, note that pseudonym and member key are included in both sides of the implication.

Checkout authenticity. For proving checkout authenticity, the chain of events is as depicted below:

```
inj-event(CustomerAcceptsReceipt(pnym, turn, com, pur, cbank, gs_pur, gs_cbank, zkpeq))
    ==> (inj-event(MerchantIssuesReceipt(turn, com, pur, cbank, gs_pur, gs_cbank, zkpeq))
      && inj-event(PSacceptsCheckout(turn, com, pur, cbank, gs_pur, gs_cbank, zkpeq))
      && inj-event(FNacceptsPayment(cbank, gs_cbank))
      && inj-event(CustomerInitCheckout(pnym, turn, com, pur,cbank, gs_pur,gs_cbank, zkpeq))).
```

In detail, the left hand side of the implication shows the event by means of which a customer accepts a receipt associated to the specified tokens. This event must be preceded by:

- The event MerchantIssuesReceipt, invoked by the merchant, proving that she has issued a receipt associated to the same tokens.
- The event PSacceptsCheckout, indicating that PS has accepted the checkout order also related to the same tokens.
- The event FNacceptsCheckout, run by FN, shows that FN has accepted the payment associated to `cbank` and its group signature, `gs_cbank`.
- The event CustomerInitCheckout, executed by the customer, which signals that the same customer has initiated a checkout, associated to the previously specified tokens.

Again, note that the pseudonym is the same in the two events launched by the customer. Moreover, the commitment, which links in zero-knowledge this pseudonym (via the member key) with both group signatures, is present in all the events, except in the one run by FN, which only includes `gs_cbank`.

Receipt authenticity. This property is also considered in the previous correspondence for checkout authenticity. Specifically, observe that all the tokens involved in event `CustomerAcceptsReceipt` are those used to generate a receipt. Moreover, these tokens come directly from the merchant who issued the receipt, given that the event `MerchantIssuesReceipt` has been invoked for the same tokens.

Finally, compared to the unforgeability properties defined at the beginning of Section 7.5, note that here the approach is in the opposite direction. Instead of proving that it is not possible to forge a turn, checkout or receipt, with ProVerif we have shown that it is only possible to create turns that follow the predefined requirements (and are thus valid and authentic).

Customer anonymity. Roughly, this property ensures that no entity should be able to identify the customer involved in a checkout process. That is, an anonymous checkout by customer A should be indistinguishable from an anonymous checkout by customer B, even for the merchant and PS. It is possible to prove this with ProVerif using its functionality for showing *observational equivalence* . In short, two processes are observationally equivalent, or indistinguishable, if an active attacker cannot tell them apart after observing their execution [42, Sec. 4.3.2][40]. In ProVerif, there are a few variations for testing different kinds of observational equivalence: strong secrecy, off-line guessing attacks and differences by terms. In our case, we will be verifying the latter. Specifically, it means that, in a given process, and given two variables X and Y within it, if the

occurrences of the former are substituted by the latter (or conversely), the result will be observationally equivalent from the original one. ProVerif can be asked to test this by specifying `choice[X,Y]`. In this case, what we do is to create two checkout objects, each one of them issued under a different member key. However, for convenience, we make the same customer process create both checkout objects (although using different member keys, as required, and thus acting as two different customers). Subsequently, we use the `choice` command to test their observational equivalence. This is included in the source code available online[7] at the moment in which the customer sends the checkout object to the merchant, using the following instructions:

```
let co1 = (msg5, com, (rk, pr, dl), turn, pur, gs_pur, cbank, gs_cbank, zkpeq) in
let co2 = (msg5, com2, (rk, pr, dl), turn2, pur2, gs_pur2, cbank2, gs_cbank2, zkpeq2) in
out (net, choice[senc(co1,kcm), senc(co2,kcm)]);
```

Where the tokens in `co1` and `co2` are created using a different member key for each of them. As a result, ProVerif informs that:

```
RESULT Observational equivalence is true (bad not derivable)
```

Which means that the attacker is not able to differentiate protocol runs in which `co1` is chosen from those in which `co2` is chosen instead.

Unlinkable turn-retrieval and checkout. This property ensures that no entity should be able to, given a checkout object, determine the turn-retrieval process that produced the turn used during checkout. The way of proving this with ProVerif is similar than the approach for showing customer anonymity. However, in this case, what we are interested in is in the specific case in which the two different checkouts have been issued by the same customer, instead of different customers. Hence, for this property, we just make use of the same `choice`, but use the same member key for both `co1` and `co2`. ProVerif's output indicates that this property is also maintained:

```
RESULT Observational equivalence is true (bad not derivable)
```

Transaction privacy against FN. This property captures the fact that FN will not learn any information concerning either the contents or the purchase or who is the related merchant. This directly follows from the fact that none of that information is included in the message received by FN, who is also a semihonest entity.

B.3 Procedural verification: computational model

In this section, we include a computational analysis of the security properties that our system preserves. As it will remain patent, this analysis complements the formal one in the sense that it deals with the possibility of the cryptographic primitives not being perfect (as is assumed in the Dolev-Yao model).

For this analysis, included in detail in [87], we start by giving game-based definitions to the security properties informally stated at the beginning of this section. Within this definitions, we use the following oracles to give the adversary additional powers when necessary.

[7] `http://www.ii.uam.es/~gnb/thesis-jdv.tgz`

- (Add clients) This oracle, written AddC, allows the adversary to add a new client. The public/private key, pseudonym, and payment information of the client will be recorded. Finally, it returns the public key and pseudonym of the client.

- (Add merchants) This oracle, written AddM, allows the adversary to add a new merchant. However, the adversary does not observe any secret information of this merchant. The public/private key of the merchant will be recorded. Finally, it returns the public key of the merchant.

- (Corrupt clients) This oracle, written CorC, allows the adversary to corrupt a client in the system, i.e., the adversary will have all information about the target client.

- (Corrupt merchants) This oracle, written CorM, allows the adversary to corrupt a merchant in the system, that is, the adversary will have all the information about the target merchant.

- (Process checkout) This oracle DoCheckout is given as input a checkout co, and if co is valid, it will process the checkout (as would be done by PS and FN.) For simplicity, we ignore AVS filters.

B.3.1 Game-based definitions

By using the previous oracles, and recovering the symbols used for depicting each token (see Table 7.1), the definitions are as follows:

Customer Anonymity. Customer anonymity requires that if a customer creates an anonymous checkout co, then no coalition of merchants, PS, and other customers should be able to determine the identity or pseudonym of the customer from co. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[\mathsf{b}, k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \mathtt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \mathtt{PSSetup}(1^k)$.
$(C_0, C_1, M, \alpha, \$) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}}, sk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{CorC}, \mathsf{AddM}, \mathsf{CorM}, \mathsf{DoCheckout})$
If C_0 or C_1 is corrupted, **return** 0.
Let (β_0, P_0) and (β_1, P_1) be the billing info and pseudonym of C_0 and C_1.
$\tau_0 \leftarrow \mathtt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P_0)[C_0(mk_0), \mathcal{A}]$
$\tau_1 \leftarrow \mathtt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P_1)[C_1(mk_1), \mathcal{A}]$
If τ_0 and τ_1 have different (risk, promo, deadline)s, **return** 0.
$co \leftarrow \mathtt{IssueAnonCheckout}(mk_b, \tau_b, \alpha, \$, \beta_b)$.
$\tilde{b} \leftarrow \mathcal{A}(co : \mathsf{DoCheckout})$.
return $\tilde{b}$.

We say that a private payment system has customer anonymity if, for any stateful PPT algorithm $\mathcal{A}$, $\left|\Pr[\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[0, k] - \Pr[\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[1, k]\right|$ is negligible in k.

Unlinkable Turn-Retrieval and Checkout. Unlinkable turn-retrieval and checkout requires that if a customer creates an anonymous checkout co, then no coalition of merchants, PS, and other customers should be able to link co to the corresponding turn-retrieval procedure. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\text{UTC}}_{\mathcal{A}}[\text{b}, k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \texttt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \texttt{PSSetup}(1^k)$.
$(C, M, \alpha, \$) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}}, sk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{CorC}, \mathsf{AddM}, \mathsf{CorM}, \mathsf{DoCheckout})$
If C is corrupted, **return** 0.
Let (β, P) be the billing info and pseudonym of C respectively.
$\tau_0 \leftarrow \texttt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P)[C(mk), \mathcal{A}]$
$\tau_1 \leftarrow \texttt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P)[C(mk), \mathcal{A}]$
If τ_0 and τ_1 have different (risk, promo, deadline)s, **return** 0.
$co \leftarrow \texttt{IssueAnonCheckout}(mk, \tau_b, \alpha, \$, \beta)$.
$\tilde{b} \leftarrow \mathcal{A}(co : \mathsf{DoCheckout})$.
return $\tilde{b}$.

We say that a private payment system has unlinkable turn-retrieval and checkout if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\left|\Pr[\mathbf{Exp}^{\text{UTC}}_{\mathcal{A}}[0, k] - \Pr[\mathbf{Exp}^{\text{UTC}}_{\mathcal{A}}[1, k]\right|$ is negligible in k.

Transaction Privacy Against FN. According to this requirement, the financial network FN should not be able to determine the detail of a customer's transaction beyond what is necessary, i.e., the customer identity and the amount of payment; in particular, the product information and the merchant identity of each transaction should be hidden from FN. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\text{TP}}_{\mathcal{A}}[\text{b}, k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \texttt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \texttt{PSSetup}(1^k)$.
$(C, M_0, M_1, \alpha_0, \alpha_1, \$) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}}, sk_{\mathsf{FN}} : \mathsf{AddC}, \mathsf{AddM})$
Let (β, P) be the payment information, and pseudonym of C.
$\tau \leftarrow \texttt{TurnRetrieval}(pk_{\mathsf{PS}}, pk_{\mathsf{FN}}, P)[C(mk), \mathsf{PS}(sk_{\mathsf{PS}})]$
$co \leftarrow \texttt{IssueCheckout}(mk, \tau, \alpha_b, \$, \beta)$.
$po \leftarrow \texttt{IssuePmtOrder}(sk_{\mathsf{PS}}, co)$.
$\tilde{b} \leftarrow \mathcal{A}(po)$.
return $\tilde{b}$.

We say that a private payment system has transaction privacy against FN if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\left|\Pr[\mathbf{Exp}^{\text{TP}}_{\mathcal{A}}[0, k] - \Pr[\mathbf{Exp}^{\text{TP}}_{\mathcal{A}}[1, k]\right|$ is negligible in k.

Turn Unforgeability. A customer should not be able to forge a valid turn that contains a risk factor or a promotion or a deadline set by his own choice. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\text{TU}}_{\mathcal{A}}[k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \texttt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \texttt{PSSetup}(1^k)$.
$\tau \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{CorC}, \mathsf{Turn}_{sk_{\mathsf{PS}}})$
If $\texttt{VerifyTurn}(pk_{\mathsf{PS}}, \tau) = 1$ and $(\tau.rk, \tau.pr, \tau.dl)$ was never observed by $\mathsf{Turn}_{\mathsf{sk_{PS}}}$
return 1; otherwise **return** 0.

We say that a private payment system has turn unforgeability if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\Pr[\mathbf{Exp}^{\text{TU}}_{\mathcal{A}}[k] = 1]$ is negligible in k.

Checkout Unforgeability. With a valid turn τ issued for a customer, no coalition of other customers, merchants, and PS should be able to forge a valid checkout co con-

taining τ. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\mathrm{TCU}}_{\mathcal{A}}[k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \mathtt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \mathtt{PSSetup}(1^k)$.
$(C, co) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}}, sk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{AddM}, \mathsf{CorC}, \mathsf{CorM}, \mathsf{Transaction})$
If co has been processed before, **return** 0.
If $\mathtt{VerifyCheckout}(co) = 0$, **return** 0.
If C has never got risk/promotion in co, but co deducts C's balance
return 1; otherwise **return** 0.

We say that a private payment system has checkout unforgeability if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\Pr[\mathbf{Exp}^{\mathrm{TCU}}_{\mathcal{A}}[k] = 1]$ is negligible in k.

Receipt Unforgeability. No coalition of customers, merchants (other than the target merchant M), and PS should be able to forge a valid receipt that looks originating from M. We describe this requirement by using the following game.

Experiment $\mathbf{Exp}^{\mathrm{RU}}_{\mathcal{A}}[k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \mathtt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \mathtt{PSSetup}(1^k)$.
$M \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{AddM}, \mathsf{CorC}, \mathsf{CorM}, \mathsf{DoCheckout})$
If merchant M is corrupted, **return** 0
$(co, rc) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{AddM}, \mathsf{CorC}, \mathsf{CorM}, \mathsf{DoConfirm})$
If the merchant M is corrupted, **return** 0
If co was queried to DoCheckout, **return** 0
If $\mathtt{VerifyReceipt}(rc, co) = 0$, **return** 0
return 1

We say that a private payment system has receipt unforgeability if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\Pr[\mathbf{Exp}^{\mathrm{RU}}_{\mathcal{A}}[k] = 1]$ is negligible in k.

Receipt Claimability. For a valid receipt from an uncorrupted customer, no other customer should be able to successfully claim the ownership of the confirmation.

Experiment $\mathbf{Exp}^{\mathrm{RC}}_{\mathcal{A}}[k]$:
$(pk_{\mathsf{FN}}, sk_{\mathsf{FN}}) \leftarrow \mathtt{FNSetup}(1^k)$.
$(pk_{\mathsf{PS}}, sk_{\mathsf{PS}}) \leftarrow \mathtt{PSSetup}(1^k)$.
$(co, rc, \pi) \leftarrow \mathcal{A}(pk_{\mathsf{FN}}, pk_{\mathsf{PS}}, sk_{\mathsf{PS}} : \mathsf{AddC}, \mathsf{AddM}, \mathsf{CorC}, \mathsf{CorM}, \mathsf{Transaction})$
If rc is never issued by Transaction, **return** 0
If the owner customer of (co, rc) is corrupted, **return** 0
If $\mathtt{VerReceiptZK}(rc, co, \pi) = 0$, **return** 0
return 1

We say that a private payment system has receipt claimability if, for any stateful PPT algorithm $\mathcal{A}$, it holds that $\Pr[\mathbf{Exp}^{\mathrm{RC}}_{\mathcal{A}}[k] = 1]$ is negligible in k.

B.3.2 Computational security proofs

We base our proofs in the security properties of the underlying building blocks. Specifically, we mainly refer to the *blindness* and *unforgeability* properties of partially blind signatures (see e.g. [156]); the *anonymity*, *misidentification security* and *framing security* properties of group signatures [132]; the *zero-knowledge* and *unforgeability* properties of zero-knowledge proof systems (see e.g. [100]); the *unforgeability* property of digital

signatures (see e.g. [109]); and the property of *protection against identity compromise* of pseudonym systems (see e.g. [141]).

Theorem 1. *Our system is customer anonymous.*

Proof. Recall that in $\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[\mathrm{b}]$, $(P_b, \tau_b, mk_b, \beta_b)$ was used (other unimportant elements were omitted). We define hybrid games as follows:
Hybrid 0. This is the actual game $\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[0]$.
Hybrid 1. It's the same as Hybrid 0 except that all ZK proofs are simulated by ZK simulator. From ZK properties for the ZK proofs, Hybrid 0 and Hybrid 1 are indistinguishable.
Hybrid 2. It's the same as Hybrid 1 except that in the first turn-retrieval (to generate τ_0), it uses $\mathtt{com} \leftarrow \mathsf{Com}(mk_1; r_{\mathtt{com}})$ instead of $\mathtt{com} \leftarrow \mathsf{Com}(mk_0; r_{\mathtt{com}})$. From the hiding property of the commitment scheme, Hybrid 1 and Hybrid 2 are indistinguishable. Note that at this moment, the two turns τ_0 and τ_1 are identically distributed.
Hybrid 3. It's the same as Hybrid 2 except that it generates *co* by running `IssueAnonCheckout` using τ_1 (and mk_0, β_0) instead of τ_0. Hybrid 2 and Hybrid 3 are indistinguishable due to blindness of the underlying partial blind signature scheme. Note that the adversary (against blindness property) first generates a key pair for the blind signature and two different messages; after signing two messages in a randomly chosen order by the game environment (for the blindness), the adversary should guess the message order (i.e., reversed or not). See [43] for more detail. We show the reduction. That is, we construct an adversary $\mathcal{B}$ breaking the blindness of the underlying partial blind scheme given an adversary distinguishing the two hybrids as follows:

> $\mathcal{B}$ runs key generation algorithm for *PBS* to generate a key pair, and sends out the pair to the outer environment. At the same time $\mathcal{B}$ works as Hybrid game (2 or 3). In particular, when $\mathcal{A}$ chooses two customers, $\mathcal{B}$ chooses $\mathtt{com}_0 = \mathsf{Com}(mk_1; r_0)$ and $\mathtt{com}_1 = \mathsf{Com}(mk_1; r_1)$ as the two messages to distinguish and sends out $(\mathtt{com}_0, \mathtt{com}_1)$ to the outer environment. Once the outer environment gives the blind signature ϱ, $\mathcal{B}$ uses it to generate *co*. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

When the signature ϱ is on $\mathtt{com}_0$, the simulation is identical to Hybrid 2; on the other hand, when ϱ is on $\mathtt{com}_1$, to Hybrid 3. Therefore, if $\mathcal{A}$ distinguishes the two hybrids with non-negligible probability, $\mathcal{B}$ also breaks the blindness property of the underlying signature with non-negligible probability.
Hybrid 4. It's the same as Hybrid 3 except that it generates *co* by running `IssueAnonCheckout` using (τ_1, mk_1, β_0) instead of (τ_1, mk_0, β_0). Hybrid 3 and Hybrid 4 are indistinguishable due to anonymity of the group signature scheme. The reduction is straightforward.
Hybrid 5. It's the same as Hybrid 3 except that it generates *co* by running `IssueAnonCheckout` using (τ_1, mk_1, β_1) instead of (τ_1, mk_1, β_0). Hybrid 4 and Hybrid 5 are indistinguishable due to semantic security of the public key encryption. The reduction is straightforward.
Hybrid 6. It's the same as Hybrid 5 except that in the first turn-retrieval (to generate τ_0), it uses $\mathtt{com} \leftarrow \mathsf{Com}(mk_0; r_{\mathtt{com}})$ instead of $\mathtt{com} \leftarrow \mathsf{Com}(mk_1; r_{\mathtt{com}})$. From the hiding property of the commitment scheme, Hybrid 5 and Hybrid 6 are indistinguishable.
Hybrid 7. It's the same as Hybrid 6 except that ZK proofs are actually done instead of using simulations. From ZK properties for the ZK proofs, Hybrid 6 and Hybrid 7 are indistinguishable. Observe that Hybrid 7 is indeed $\mathbf{Exp}^{\mathrm{CA}}_{\mathcal{A}}[1]$, which concludes the proof. □

Theorem 2. *Our system has the property of unlinkable turn-retrieval/checkout.*

Proof. The proof is the same as showing indistinguishability of Hybrid 3 and 4 in the previous proof. □

Theorem 3. *Our system has transaction privacy against* FN.

Proof. Our system simply drops all information about α when PS creates a payment order, and this guarantees transaction privacy against FN, information-theoretically. □

Theorem 4. *Our system satisfies turn unforgeability.*

Proof. Turn unforgeability simply follows from unforgeability of the partial blind signature scheme. □

Theorem 5. *Our system satisfies checkout unforgeability.*

Proof. Checkout unforgeability follows from soundness of ZK proofs. In particular, $\tau.\mathtt{com}$ should be the commitment to mk_i due to the soundness of the proof in the turn-retrieval. Moreover, the soundness of ZK proof ψ in the checkout object makes sure that all of $\tau.\mathtt{com}$, ϱ_α and $\varrho_{\tilde{\beta}}$ use the same member key mk_i. □

Theorem 6. *Our system satisfies receipt unforgeability.*

Proof. Receipt unforgeability holds immediately from unforgeabilty of underlying the digital signature scheme. □

Theorem 7. *Our system satisfies receipt claimability.*

Proof. Receipt unforgeability holds immediately from security against framing attacks of the underlying group signature scheme. □

APPENDIX C

Using libgroupsig

In this section we explain how to configure, compile and make use of the library through a few simple code snippets for the main actions. `libgroupsig` requires glib (version 2.33 or compatible), openssl (version 1.0.1e-2 or compatible) for hashing functions, libgmp (version 2:5.0.5 or compatible) and the PBC library[1] (0.5.12 or compatible). Therefore, in order to use it, these libraries must be installed in the system. Also, the library uses the GNU build system[2]. Thus, in order to check the environment and generate the proper compilation scripts, the `configure` script must be run. Afterwards, the library is compiled with `make` and optionally installed with `make install`. A minimal set of auxiliary tools (located under the *tools* folder) for testing the library may be compiled with `make check`. Below we include a few code snippets, mostly extracted from the mentioned *tools* programs, showing some of the main functionality of the library. A detailed API documentation is available within the library's home page at Bitbucket[3]

Group creation The code snippet in Figure C.1 shows how to create a group. Specifically, it creates the group and manager keys and an empty GML, using predefined configuration values for each supported group signature scheme. The initial `groupsig_init` call sets up library-wide structures (currently, it seeds random number generators). Subsequently, the group and manager keys, and GML are initialized. Finally, the group is created by filling up the initialized cryptographic tokens and setting scheme-wide data structures (e.g., PBC data structures for pairing based group signature schemes).

Adding group members This operation typically requires some precomputation by the new member and a finalization by the group manager. Thus, we have divided it accordingly. The result of each operation may just be transmitted over the network. However, for brevity, we include it in the snippet in Figure C.2 as part of the same program. After successfully adding a new member, the GML (required parameter to the group manager side of the process) will be updated with the new member information.

Signing messages and signature verification Figure C.3 shows the process for creating a group signature. It is worth emphasizing the last parameter which, in case of being other than `UINT_MAX`, specifies that the random number generator must be re-

[1] `http://crypto.stanford.edu/pbc/`
[2] `http://en.wikipedia.org/wiki/GNU_build_system`
[3] `https://bitbucket.org/jdiazvico/libgroupsig/`

```
1  /* Initialize environment */
2  if(groupsig_init(time(NULL)) == IERROR) { return IERROR; }
3
4  /* Set group signature scheme configuration parameters. */
5  if(cfg->scheme == GROUPSIG_KTY04_CODE) {
6    KTY04_CONFIG_SET_DEFAULTS((kty04_config_t *) cfg->config, key_format);
7  } else if(cfg->scheme == GROUPSIG_BBS04_CODE ||
8            cfg->scheme == GROUPSIG_CPY06_CODE) {
9    CPY06_CONFIG_SET_DEFAULTS((cpy06_config_t *) cfg->config, key_format);
10 }
11
12 /* Initialize the group key, manager key and GML variables */
13 if(!(mgrkey = groupsig_mgr_key_init(cfg->scheme))) { return IERROR; }
14 if(!(grpkey = groupsig_grp_key_init(cfg->scheme))) { return IERROR; }
15 if(!(gml = gml_init(cfg->scheme))) { return IERROR; }
16
17 /* ''Construct'' the group */
18 if(groupsig_setup(cfg->scheme, grpkey, mgrkey, gml, cfg) == IERROR) {
19   return IERROR;
20 }
```

Figure C.1: Group creation.

```
1  /* Initialize member key structure */
2  if(!(memkey = groupsig_mem_key_init(cfg->scheme))) { return IERROR;}
3
4  /* Member side of join */
5  if(groupsig_join_mem(memkey, grpkey) == IERROR) { return IERROR; }
6
7  /* Group manager side of join */
8  if(groupsig_join_mgr(gml, memkey, mgrkey, grpkey) == IERROR) {
9    return IERROR;
10 }
```

Figure C.2: Addition of new group members.

seeded using the specified value. Verification of a group signatures is shown in Figure C.4.

Opening signatures With the open function, the real identity of the signer of a group signature is obtained. It requires the group signature itself and the group membership list besides, of course, the group manager key. Figure C.5 shows how to call the function. Once obtained the identity of the signer, its member key may be revoked by including it in a CRL which, in turn, may be used to trace dishonest users, and even made public.

Other functions The previous functions represent the core of group signature schemes. However, specific schemes may implement additional functionality, like tracing dishonest users and claiming group signatures. For implementing these functions, the library provides handlers following the same style than the already introduced ones. Also, miscellaneous functionality is provided for importing, exporting and copying keys and signatures.

```
1  /* Initialize the group signature object */
2  if(!(sig = groupsig_signature_init(scheme))) {
3    fprintf(stderr, "Error: failed to initialize the group signature.\n");
4    return IERROR;
5  }
6
7  /* Sign the message: setting the seed to UINT_MAX forces to
8     get a new pseudo random number for this signature instead
9     of using a pre-fixed random number. */
10 if(groupsig_sign(sig, msg, memkey, grpkey, UINT_MAX) == IERROR) {
11   fprintf(stderr, "Error: signing failure.\n");
12   return IERROR;
13 }
```

Figure C.3: Issuing group signatures.

```
1  /* Verify group signature */
2  if(groupsig_verify(&bool, sig, msg, grpkey) == IERROR) {
3    fprintf(stderr, "Error: verification failure.\n");
4    return IERROR;
5  }
6  if(!bool) { fprintf(stdout, "WRONG signature.\n"); }
7  else { fprintf(stdout, "VALID signature.\n"); }
```

Figure C.4: Verifying group signatures.

```
1  /* Open group signature */
2  if((rc = groupsig_open(id, sig, grpkey, mgrkey, gml)) == IERROR) {
3    fprintf(stderr, "Error opening signature.\n");
4    return IERROR;
5  }
```

Figure C.5: Opening group signatures.

C.1 Extending `libgroupsig`

The library includes a script, named `libgroupsig.sh` and located under the `tools` directory, which allows the automated creation of the skeleton of a new group signature scheme. This option is invoked with the command `./libgroupsig.sh addscheme <scheme name>`, and creates a new subdirectory `$libroot/groupsig/<scheme name>` containing this "empty" skeleton and updating a few library wide data structures. After running this command, the programmer would have to implement the actual functionality within the files created inside the new subdirectory (e.g. `setup.c`, `sign.c`, etc.).

Index